MÁSTER EN TRATAMIENTO ESTADÍSTICO-COMPUTACIONAL DE LA INFORMACIÓN

TRABAJO FIN DE MÁSTER

Analysis of the Transformer Architecture and application on a Large Language Model for mental health counseling

Andrés Herencia López-Menchero

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

&

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

Tutores: Mario Vega Barbas & Ignacio Villanueva Díez

Madrid, Junio 2024

MÁSTER EN TRATAMIENTO ESTADÍSTICO-COMPUTACIONAL DE LA INFORMACIÓN

TRABAJO FIN DE MÁSTER

Análisis de la arquitectura Transformer y aplicación en un Gran Modelo de Lenguaje para asesoramiento en salud mental

Andrés Herencia López-Menchero

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

&

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

Tutores: Mario Vega Barbas & Ignacio Villanueva Díez

Agradecimientos

Con este Trabajo Fin de Máster (TFM), culmino mi trayectoria académica. Con él, he querido reflejar todo el conocimiento que he adquirido a lo largo de estos años, desde el grado hasta el máster, no solo desde un punto de vista académico si no de desarrollo personal y profesional.

Es un trabajo que ha sido complicado, pues analizar cada pequeño detalle de esta arquitectura, así como su implementación en un estado del arte tan novedoso ha sido una tarea titánica. Aunque mentiría si no dijera que, a su vez, ha sido muy gratificante. Y este camino se ha hecho más fácil por aquellas personas que me han acompañado en este proceso, a los que quería expresar mi más profundo agradecimiento.

A mi familia, que incondicionalmente me ha apoyado, y siempre me ha incentivado a hacer aquello que me hace feliz. A mi madre, a mi padre, a mi hermana. Gracias. Los sueños se hacen más fáciles cuando ves a quienes más quieres apoyándote a tu lado.

A mis amigos, tanto los que han estado conmigo desde siempre como aquellos que he recogido a lo largo de esta etapa, siendo afortunadamente, muchos y valiosos. Por vuestro apoyo, y por vuestro cariño, mil gracias.

A mis compañeros de piso, que me muestran una parte más del mundo y de mi mismo, mostrándome una perspectiva diferente del mundo que nos rodea. Y como no, a mis hermanos italianos: "ci vediamo presto".

A mis tutores. A Mario Vega, por su increíble implicación y apoyo, con sus valiosos consejos y su increíble guía en este vertiginoso proceso. Creo que ambos hemos aprendido mucho durante este camino. Y también a Ignacio Villanueva, por su dedicación e ideas, aportando ese granito de experiencia más que apreciado.

En general, solo tengo palabras de júbilo tras la realización de este proyecto. Aunque mi camino en este máster acabe, mis ganas de seguir formándome y aprendiendo no han cesado aún. Espero que este trabajo aporte, aunque sea un poco, en **Transformar** el mundo en un lugar mejor.

Abstract

The rapid advances of generative Artificial Intelligence (AI) have marked a milestone in the Natural Language Processing (NLP) field. Specifically, Transformer models have revolutionized the state-of-the-art due to their great effectiveness and efficiency in several tasks, both general and specific. Thus, this work explores the Transformer architecture and its application to Large Language Models (LLMs) through Parameter-Efficient Fine Tunning (PEFT) techniques, aiming to create a conversational model for mental health counseling. The study provides a detailed explanation of the architecture, including its theoretical and mathematical foundations. The fine-tuning process uses a novel state-of-the-art technique known as Low Rank Adapters (LoRA). Subsequently, the performance is evaluated by comparing the original with the fine-tuned model to verify the adaptation performance. Some conclusions are extracted at the end of the document, highlighting the most important advantages and disadvantages of the applied methodology.

Keywords: Large Language Model (LLM), Deep Learning (DL), Parameter-Efficient Fine Tunning (PEFT), Low Rank Adapters (LoRA), LLaMA, Transformer.

Resumen

Los rápidos avances en Inteligencia Artificial (IA) generativa ha supuesto un hito en el campo del Procesamiento del Lenguaje Natural (PLN). Concretamente, los modelos Transformer han revolucionado el estado del arte a través de su gran eficacia y eficiencia en gran variedad de tareas, tanto generales como específicas. Así, este trabajo explora la arquitectura Transformador y su aplicación a Modelos del Lenguaje Grandes (MLG) a través de técnicas de sobre-entrenamiento, para crear una herramienta que sirva para asesorar y aconsejar en el ámbito de la salud mental. El estudio explica detalladamente la arquitectura, incluyendo los fundamentos teóricos y matemáticos. El proceso de sobre-entrenamiento hace uso de una técnica novedosa en el estado del arte, conocida como LoRA (Low Rank Adapters). Posteriormente, se evalúa el rendimiento de este modelo con respecto al original, comprobando la efectividad de la adaptación. Algunas conclusiones son extraídas al final del documento, destacando las ventajas e inconvenientes de la metodología aplicada, así como futuras posibles líneas de investigación del proyecto.

Palabras clave: Large Language Model (LLM), Aprendizaje Profundo, Parameter-Efficient Fine Tunning (PEFT), Low Rank Adapters (LoRA), LLaMA, Transformador.

Contents

List of Figures

Acronyms

ADALINE ADAptative LINear Element. 4

AI Artificial Intelligence. 1, V

ANN Artificial Neural Network. 3, 31, 32, VII, X

B-BP Bidirectional Backward Propagation. 11

BART Bidirectional and Auto-Regressive Transformer. 31

BERT Bidirectional Encoder Representation from Transformers. 30, 41, 43, 49

BPTT Back-Propagation Through Time. 9

CNN Convolutional Neural Network. 31

DL Deep Learning. 3, 7, 26, 39, V

DPR Dense Passage Retrieval. 30

FNN Feed-Forward Neural Network. 6, 7, 13, 18, 21, X

GLU Gated Linear Unit. 24

GQA Grouped Query Attention. 24

GRU Gated Recurrent Unit. 9, 10, VI–VIII, XI

LLaMA Large Language Model Meta AI. 1, 23, 24, 33, 37, 39, 46, 48–50, V

LLM Large Language Model. 1–3, 23, 26–29, 33–35, 39, 49, V, IX, X

LMS Least-Mean-Square. 7

LoRA Low Rank Adapters. 32–34, 39, 43, 44, 49, V

LSTM Long short-term memory. 9, 10, V–VIII, XI

METEOR Metric for Evaluation of Translation with Explicit Ordering. 41–43, 49

MHA Multi-Head Attention. 14, 15, 17, 19–22, X

MIPS Maximum Inner Product Search. 30

MLP Multi-Layer Perceptron. 5, 6

MSE Mean Squared Error. 7

NLP Natural Language Processing. 7, V, XII

PEFT Parameter-Efficient Fine Tunning. 26, 32, 49, V

QLoRA Quantized Low Rank Adapters. 32, 34, 49

RAG Retrieval-Augmented Generation. 26, 27, 29, 34, X

RBP Recurrent Back-Propagation. 9

ReLU Rectified Linear Unit. 21, 24

RMS Root Mean Square. 24

RNN Recurrent Neural Network. 7–13, VIII, X, XI

RoPE Rotary Positional Embeddings. 24

RTRL Real-Time Recurrent Learning. 9

RU Recurrent Units. 9

s2s Sequence to Sequence. 9, X, XI

SMT Statistical Machine Translation. 10, 12, 40, III, IV, XI

SwiGLU Swish-Gated Linear Unit. 24

TL Transfer Learning. 31, X

TSA Time Series Analysis. 20

Chapter 1

Introduction

1.1 Context

Artificial Intelligence (AI) has solved new classification, prediction and generation problems. Specifically, Transformer models have achieved groundbreaking progress in the generative AI field. This architecture can be used to build Large Language Models (LLMs) which can be adapted to different contexts.

The present project aims to modify an existing LLM to provide expert knowledge in the mental health counseling domain. This model will be tailored to provide support and guidance in mental health scenarios. Subsequently, its performance will be compared with the original model to evaluate the effectiveness of the fine-tuning process. State-of-the-art LLM customization techniques will be reviewed, as well as the priorly introduced architectures before the irruption of the Transformer model.

1.2 Objectives

This project aims for the following objectives:

1. Provide a comprehensive analysis of the Transformer architecture, detailing mathematically and theoretically each layer.
2. Handle and manage an open-source language model, LLaMA.
3. Fine-tune the model for adapting to the specific domain of mental health counseling.
4. Assess the model's performance in delivering relevant and empathetic mental health support and establish a comparison with the answers provided by humans and the base non-fine-tuned model.

1.3 Resources

The available resources can be classified into three main categories: computational, data, and mathematical.

- **Computational resources** refer to the necessary hardware and software. For building, fine-tuning, and evaluating LLMs, Python and its libraries are used. Due to the computational complexity of these models, virtual machines provided by the Google Cloud Platform (GCP)[1] and Google Colab[2] are employed.
- **Mental health conversation data** is gathered from diverse sources, then transformed and loaded into public repositories.
- **Mathematical tools** include papers, articles, and books that explore state-of-the-art techniques and statistical methods learned during the Master's Degree, along with other online courses and materials focused on neural networks, in-context learning, and fine-tuning models. These references can be consulted in the Bibliography.

1.4 Document structure

The document is divided into seven chapters, which consist of:

1. **Introduction:** Introduces the thesis context, objectives, and resources.
2. **Theoretical Background:** Details the foundational architectures, methods, and technologies that serve to build the Transformer architecture.
3. **Transformer architecture:** Includes a detailed review of this model, detailing the most significant advancements, focusing on the attention mechanisms.
4. **State-of-the-art:** Encompasses the novel LLM customization techniques aside from the enabling technologies to produce this project.
5. **Implementation and Development:** Presents the steps required to perform model fine-tuning, besides its evaluation.
6. **Results:** Shows the obtained results, with their corresponding interpretations.
7. **Conclusions:** Summarizes the evaluations obtained in the work, presenting insightful conclusions and future lines of action.

[1] https://cloud.google.com/
[2] https://colab.research.google.com/

Chapter 2

Theoretical background

Efficient and accurate text transformation tasks have long been a goal in machine design. The Transformer architecture has addressed this challenge. However, its development has relied on prior architectures. This chapter outlines the architectures enabling the emergence of the Transformer and upcoming LLM.

2.1 Neural network concept

Transformers are fundamentally based on Deep Learning (DL) models, so knowing about neural networks is mandatory. A neural network is a computational architecture used in statistical learning tasks. Every neural network is characterized by three elements: (i) the neurons; (ii) the connections; and (iii) the layers. There are three main layers: the input layer, the output layer, and the hidden layers (can be more than one). [62]

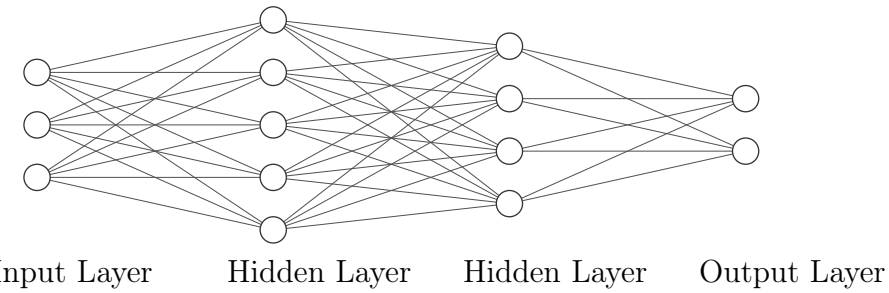


Figure 2.1: Artificial Neural Network (ANN) structure diagram.

The neurons are described as the minimal computational unit in a neural network. Each neuron computes the output as a weighted sum of the input data processed by a non-linear function. The connections module the weights of each neuron's relations. In Figure 2.1, a neural network with an input layer with three neurons, two hidden layers with five and four neurons, and an output layer with two neurons is shown.

2.1.1 Simple Perceptron

The most basic neural network architecture is known as simple perceptron. It has a unique neuron, working as a binary classification machine for linearly separable problems. [25] Thus, the input, output, and hidden layers are the same.

Input data is summed and weighted via the ADAptative LINear Element (ADALINE). This component adds a bias term, denoted by b. Then, this data is processed by an activation function, a generally monotonous, continuous, bounded, and non-linear application used to model complex relationships between input and output data.

In the specific case of simple perception (which acts as a linear binary classification machine), the sign function is used,

$$\operatorname{sgn}(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases}$$

but other functions are used in different contexts and models. The most important ones are explained below.(observe Figure 2.2). [54]:

- **Sigmoid-shaped functions**: $\sigma(x) = \frac{1}{1-e^{-}\alpha x}$, with $\alpha \in \mathbb{R}$, and being x a scalar number.
- **Hyper parabolic tangent function**: $\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$.
- **Rectified Linear Unit**: $\operatorname{ReLU}(x) = \max(0, x)$ used in other contexts and neural network types.
- **Softmax function**: Used in multi-classification contexts, $\text{softmax} = \sigma(\mathbf{z_j}) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$, for $j = 1, \ldots, K$, being $\mathbf{z}$ a $\mathbb{R}^K$-dimensional vector.

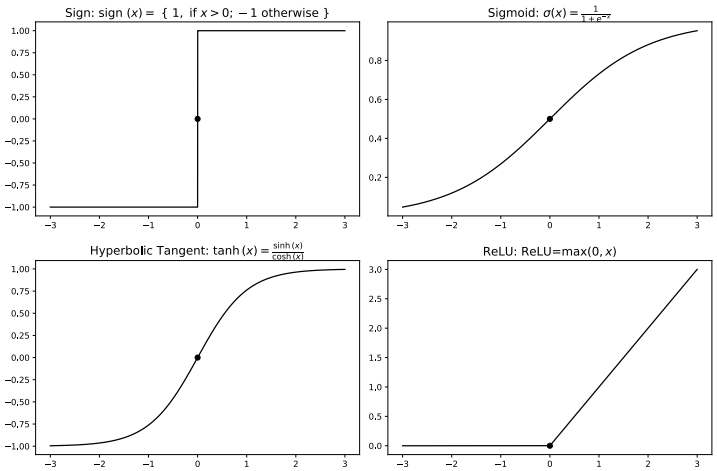


Figure 2.2: Activation function examples.

Output computation

Let $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ the input vector and $\mathbf{w} = (w_1, w_{,2}, \ldots, w_N)$ the weights vector. Then, the output can be computed as follows:

$$y = \text{sgn} \sum_{i=1}^{N} w_i x_i + b \tag{2.1}$$

The architecture diagram can be shown in Figure 2.3.

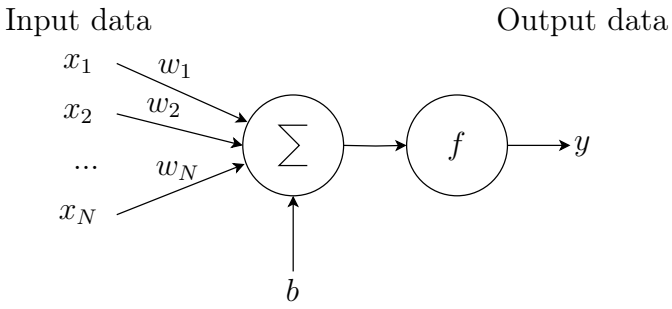


Figure 2.3: Simple perceptron architecture.

Training

In the training process, the aim is to obtain a weights vector such that all the produced outputs (based on the supervised training set, $y^d = \{y_1^d, y_2^d, \ldots, y_n^d\}$) are well classified. For this purpose, a learning rule is followed, governed by a discrete-time dynamic system: [22]

$$\mathbf{w_{t+1}} = \mathbf{w_t} + \eta(y^d - y)\mathbf{x} \tag{2.2}$$

where η is the learning rate, a parameter that modulates the speed at which the weights are updated.

Since the output is binary, the system will converge when the expected output equals the computed output, $y^d = y$ [19]. Therefore, this method can only be applied to linear-separable classification problems.

To deal with this constraint, more neurons and layers can be aggregated to capture complex relationships. [49] These are arranged in the three layers mentioned above (section 2.1), and they build a machine known as a Multi-Layer Perceptron (MLP).

2.1.2 Multi-Layer Perceptron (MLP)

In the Multi-Layer Perceptron architecture, the output of each neuron in a layer is computed by the weighted sum of the neurons' outputs in the previous layer, applied to an activation function. This process is repeated for each layer. Observe Figure 2.4

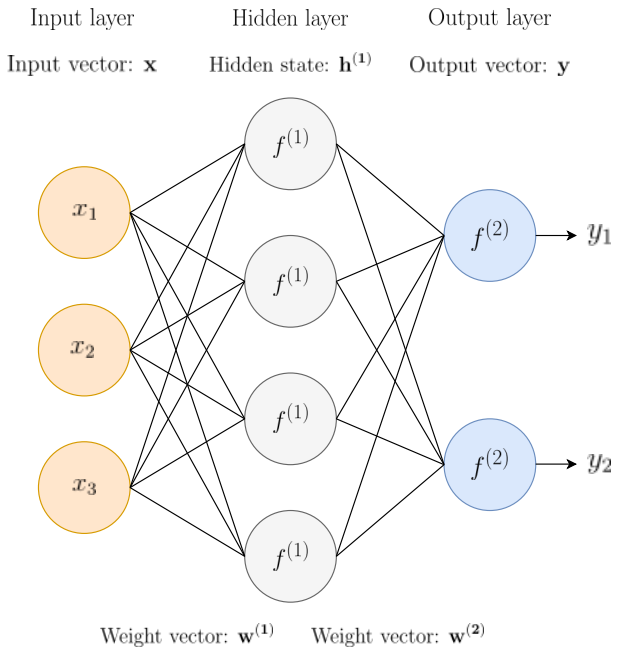


Figure 2.4: FNN basic structure.

Output computation

Assume a MLP of L layers. $a^{(l)}$ is defined as the activation term of layer l, $a^{(l)} = f^{(l)}(\mathbf{w}^{(l)}a^{(l-1)} + b^{(l)})$ where the first term is $a^{(1)} = f^{(1)}(\mathbf{w}^{(1)}\mathbf{x} + b^{(1)})$. Note that it is the expression obtained in the equation 2.1 but generalized for all the neurons in the first layer.

The expression in the output layer can be obtained by:

$$y = f^{(L)}(\mathbf{w}^{(L)}a^{(L)} + b^{(L)}) \tag{2.3}$$

Equivalently:

$$y = f^{(L)}(\mathbf{w}^{(L)} \cdot f^{(L-1)}(\mathbf{w}^{(L-1)} \dots (\mathbf{w}^{(1)} \cdot \mathbf{x} + b^{(1)}) + b^{(2)}) + \dots + b^{(L-1)}) + b^{(L)})$$

where $f^{(l)}$ is the activation function of the l-layer, $\mathbf{w^{(l)}} = \{\mathbf{w_1^{(l)}}, \mathbf{w_2^{(l)}}, \dots, \mathbf{w_M^{(l)}}\}$ is the set of the l-layer weight vectors for the M neurons and $b^{(l)} = (b_1^{(l)}, b_2^{(l)}, \dots, b_M^{(l)})$ is the l-layer bias vector.

Training

This model is trained following the back-propagation rule, a LMS-based (Least Mean Square) optimization method. The objective is no longer to properly classify every point, but to minimize the error at the output, ridding the restriction that the outputs be binary.

Hence, a cost function is defined, typically involving the cross-entropy loss function (for classification tasks) or the Mean Squared Error (MSE) (for regression tasks). Using the MSE as example:

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - y_i^d)^2 \tag{2.4}$$

The minimization problem will be to find the argument which minimizes the cost function: $\mathbf{w}^* = \arg\min E(\mathbf{w})$.

This architecture is trained following an iterative scheme based on the gradient descent [2] method, which searches in that direction where there may be a local minimum. Thus, the weight vector is updated following the expression:

$$\mathbf{w_{t+1}} = \mathbf{w_t} - \eta \cdot \left.\frac{\partial E}{\partial \mathbf{w}}\right|_{w_t} \tag{2.5}$$

When the network is computed backward, this term is effectively obtained. Since the perceptron uses this rule for training, this scheme is also known as Feed-Forward Neural Network (FNN). Those models that use more than one hidden layer are known as Deep Learning models.

The FNN has the problem that each input vector is processed independently. So, they cannot capture information about temporal, ordinal, or sequential relationships. This is crucial for text transformation tasks, where the data are essentially word sequences related to preceding and following words.

Therefore, the architecture was adapted so that the network sends information to and from any of its hidden layers in the training phase. This adaptation founded a new neural network scheme, the Recurrent Neural Network (RNN). [39]

2.2 Recurrent Neural Networks (RNNs)

RNN is a popular architecture for Natural Language Processing (NLP) tasks, such as translation, text processing, or sentiment analysis.

For each discrete time interval t, the system has a hidden state $\mathbf{h_t}$. The hidden state $\mathbf{h_t} = (h_1, h_2, \ldots, h_M)$, consists of M hidden units which represents the vector of all $a^{(l)}$ terms of the hidden layers for the time step t. Thus, the system is fed back with each of the previous and subsequent states, in addition to all inputs. The vector $\mathbf{y} = (y_1, y_2, \ldots, y_P)$ [56] is produced as output. Observe Figure 2.5.

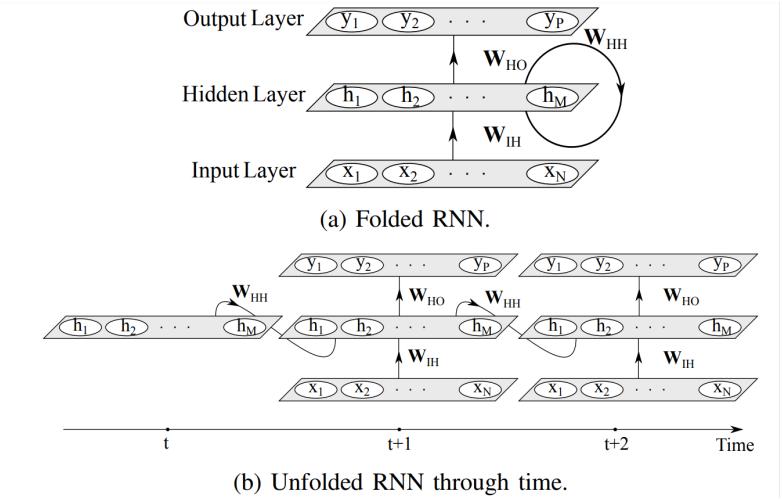


Figure 2.5: RNN architecture. [56]

2.2.1 Output computation

The hidden state $\mathbf{h_t}$ is computed as a function of the previous hidden state $\mathbf{h_{t-1}}$ and the current input $\mathbf{x_t}$.

$$\mathbf{h_t} = f_H\left(\mathbf{W_{HH}h_{t-1}} + \mathbf{W_{IH}x_t} + \mathbf{b_h}\right) \tag{2.6}$$

Thus, the output calculation at time step t is obtained from:

$$\mathbf{y_t} = f_O\left(\mathbf{W_{HO}h_t} + \mathbf{b_o}\right) \tag{2.7}$$

where $\mathbf{W_{HH}}$ is the weights matrix for the connections between the previous hidden state $\mathbf{h_{t-1}}$ and the subsequent (current) hidden state $\mathbf{h_t}$; $\mathbf{W_{IH}}$ is the weights matrix for the connections between the current input $\mathbf{x_t}$ and the hidden state. $\mathbf{W_{HO}}$ is the weights matrix for the connections between the last layer of the hidden state and the output layer. Finally, $\mathbf{b_h}$ and $\mathbf{b_o}$ are the bias vectors for the hidden state and the last layer; and f_H and f_O are the activation functions of the hidden state and the output layer, respectively.

2.2.2 Training

Several training methods are used in this architecture. Most of them are supervised learning methods based on the gradient descent algorithm: Real-Time Recurrent Learning (RTRL), Recurrent Back-Propagation (RBP), or Back-Propagation Through Time (BPTT). [9] The latter is probably the most used one.

The RNN conventional architecture presents significant challenges, notably the gradient vanishing and explosion problems. As detailed in [57], small gradients diminish exponentially as they propagate back through time, leading to the vanishing or fading gradient issue. Conversely, very high gradients can increase exponentially, resulting in an explosion problem. Both scenarios hinder the effective training of the network.

To solve this problem, several recurrent architectures were raised, such as Long short-term memory (LSTM) cells and Gated Recurrent Unit (GRU) cells (these architectures can be consulted on appendices C and D, respectively). These models tried to avoid the gradient vanishing problem using memory mechanisms. However, the most relevant architecture that has led to improved performance in machine translation tasks was the encoder-decoder.

2.3 Encoder-decoder architecture

This architecture (also known as Sequence to Sequence (s2s) model) consists of three parts: (i) the encoder; (ii) the encoding vector; and, (iii) the decoder. See Figure 2.6.

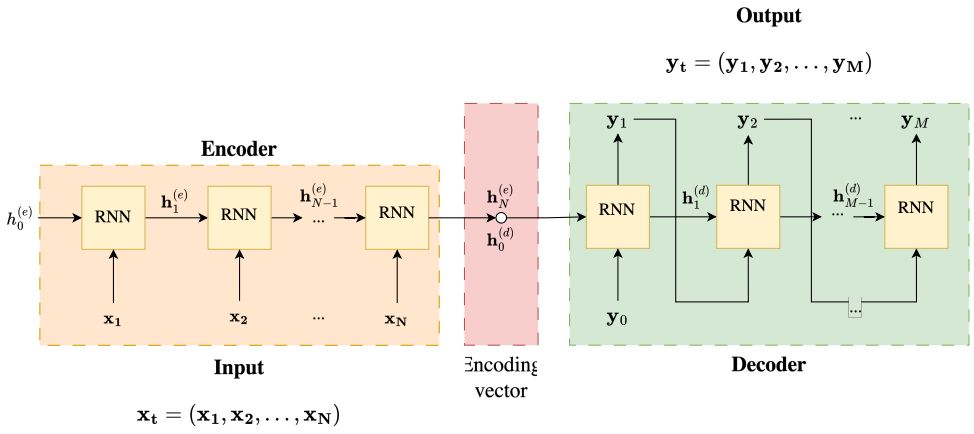


Figure 2.6: Sequence to Sequence (s2s) architecture for time step t.

- The **encoder** is composed of N Recurrent Units (RU) (i.e., RNNs, LSTM or GRU cells) where each accepts a single element of the input sequence, collects information for that element and propagates it forward.

- The **encoder vector** is the final hidden state produced by the encoder. This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.
- The **decoder** is a stack of M recurrent units where each predicts an output y_t at a time step t. Each recurrent unit accepts a hidden state from the previous unit and produces an output and a hidden state.

This architecture was typically used for question-answering problems (a practical approach to this problem lies in a chatbot application) or translation systems (English to French translation, for instance). [31] So, input and output are the words from the source and target sentences. In fact, this architecture is an evolution of the classical Statistical Machine Translation (SMT) system (which can be found described in Appendix B).

Since the input is usually a sentence, represented by vectors, from now on forward input will be considered as $\mathbf{x_t} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N})$, meanwhile the output will be $\mathbf{y_t} = (\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_M})$.

2.3.1 Output computation

If the encoder block output is denoted as $\text{RNN}^{(e)}(\cdot)$, and the decoder output is denoted as $\text{RNN}^{(d)}(\cdot)$, then the model can be expressed in the following terms [43]:

- **Encoder output**: corresponds to the last hidden state of the encoder, computed by the expression:

$$\mathbf{h_t^{(e)}} = \text{RNN}^{(e)}(\mathbf{x_t}, \mathbf{h_{t-1}^{(e)}}) \tag{2.8}$$

  Note that $\mathbf{h_0^{(e)}} = 0$. In MT context, $\mathbf{x_N}$ is always a <eos> (*end of string*) indicator.

- **Decoder output**: corresponds to the last hidden state of the decoder, computed by the expression:

$$\mathbf{y_t} = \text{softmax}(\mathbf{W_{HO}}\mathbf{h_t}^{(d)} + \mathbf{b_O}). \tag{2.9}$$

$$\mathbf{h_t}^{(d)} = \text{RNN}^{(d)}(\mathbf{y_{t-1}}, \mathbf{h_{t-1}^{(e)}}) \tag{2.10}$$

  In MT context, y_0 is always a <bos> (*beginning of string)* indicator.
  Since $(\mathbf{W_{HO}}\mathbf{h_t}^{(d)} + \mathbf{b_O})$ returns a multi-dimensional vector, softmax function $(f : \mathbb{R}^M \to \mathbb{R})$ must be applied.

NOTE: despite the encoder and decoder being described with the RNN architecture, derived architectures such that GRU or LSTM cells can be used.

2.3.2 Training

Encoder-decoder architecture can be trained using Back-Propagation Through Time (BPTT), as is common with other architectures. Regularization techniques such as mini-batch processing [35], gradient clipping [68], and early stopping are often utilized to ease training and prevent overfitting. Another common algorithm is the Bidirectional Backward Propagation (B-BP) method, as explained in the following section. [53]

This architecture encounters multiple challenges, including information bottlenecks (resulting from fixed-length vector representations), difficulties in managing long sequences, and issues with aligning input and output sequences, which stem from the architecture's origins in Statistical Machine Translation (SMT) models, among other problems. Attention mechanisms alleviate these issues by enabling selective focus on different parts of the input sequence, handling long sequences effectively, and learning soft alignments.

2.4 Architectures with attention mechanisms

Attention mechanisms allow models to focus on a specific text part, avoiding memory problems. Conceptually, it can be understood as a resource allocation scheme, so, a priori, it can solve the information bottleneck problem mentioned above.

2.4.1 RNNSearch

RNNsearch responds to an encoder-decoder architecture based on Bidirectional Recurrent Neural Networks (BiRNN) in the encoder and an attention-based decoder.

Encoder

The BiRNN for a time step t does not exclusively take the previous and current inputs but, computes the hidden states by taking all the future and previous input data. It is trained in two directions: backward, from $\mathbf{x_N}$ to $\mathbf{x_1}$, denoting the hidden states as $\overleftarrow{\mathbf{h_t}}^{(e)} = \{\overleftarrow{\mathbf{h_1}}^{(e)}, \overleftarrow{\mathbf{h_2}}^{(e)}, \ldots, \overleftarrow{\mathbf{h_N}}^{(e)}\}$; and forward, from $\mathbf{x_1}$ to $\mathbf{x_N}$ denoting the hidden states as $\overrightarrow{\mathbf{h_t}}^{(e)} = \{\overrightarrow{\mathbf{h_1}}^{(e)}, \overrightarrow{\mathbf{h_2}}^{(e)}, \ldots, \overrightarrow{\mathbf{h_N}}^{(e)}\}$. This process is known as a Bidirectional Backward Propagation (B-BP).

When all the hidden states are computed in each direction, an annotation for each $\mathbf{x_i}$ is obtained concatenating both priorly computed hidden states: $\overline{\mathbf{h_i}}^{(e)} = [\overrightarrow{\mathbf{h_i}}^{(e)}; \overleftarrow{\mathbf{h_i}}^{(e)}]^T = \mathbf{h_i}^{(e)}$. The annotations are, in fact, the codification of the inputs and at the same time, the

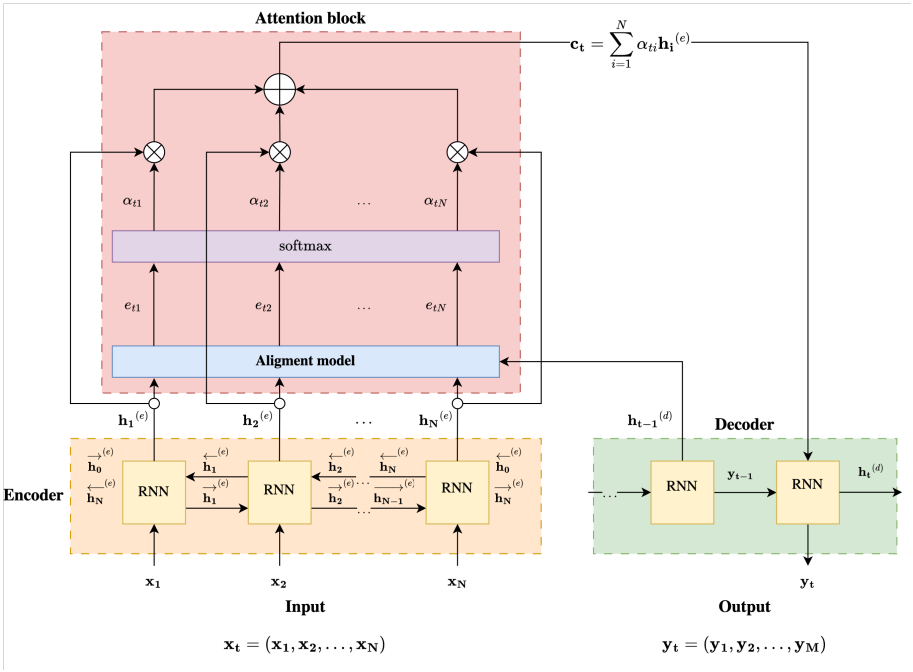


Figure 2.7: *RNNsearch* architecture. Decoder is shown only for time step t.

hidden states of the bidirectional RNN [4]:

$$(\mathbf{h_1}^{(e)}, \ldots, \mathbf{h_N}^{(e)}) = \text{BiRNN}^{(e)}(\mathbf{x_1}, \ldots, \mathbf{x_N}) \tag{2.11}$$

When hidden states are computed, this information is sent to the decoder.

Decoder

The decoder is built with an attention block and a unidirectional RNN. In SMT problems, the attention block captures the text context in a vector, representing the relationship between the current output vector and each input term. So, at a time step t, the context vector $\mathbf{c_t}$ is computed as a weighted sum of the weighted annotations $\mathbf{h_i}^{(e)}$ as follows:

$$\mathbf{c_t} = \sum_{i=1}^{M} \alpha_{ti} \mathbf{h_i}^{(e)} \tag{2.12}$$

The attention weight α_{ti} of the annotation $\mathbf{h_i}^{(e)}$ is computed by

$$\alpha_{ti} = \text{softmax}(e_{ti}) = \frac{\exp(e_{ti})}{\sum_{k=1}^{M} \exp(e_{tk})} \tag{2.13}$$

where e_{ti} is described as the alignment model, $e_{ti} = a(\mathbf{h_{t-1}}^{(d)}, \mathbf{h_i}^{(e)})$, being $a(\cdot)$ a learnable function. This term reflects the importance of the encoder's annotation $\mathbf{h_i^{(e)}}$ to the next decoder hidden state $\mathbf{h_t^{(d)}}$ according to the decoder state $\mathbf{h_{t-1}^{(d)}}$ [44]. It is typically param-

eterized by a small neural network (often a FNN). The alignment model scores the match between an input position i and the current time step t, guiding the decoder's attention to the most relevant parts of the input sequence when generating each word in the output sequence.

After that, the RNN outputs the most probable symbol $\mathbf{y_t}$ at time step t as:

$$p(\mathbf{y_t}|\mathbf{y_1}, \ldots, \mathbf{y_M}; \mathbf{x_t}) = \text{RNN}^{(d)}(\mathbf{c_t}) \tag{2.14}$$

As in the encoder-decoder architecture, any recurring unit can be used to build this machine.

This approach distributes the information from the source sentence across the entire sequence, rather than encoding it all into a fixed-length vector through the encoder. Meanwhile, the decoder can selectively access this information at each time step. This formulation allows the neural network to concentrate on pertinent elements of the input rather than irrelevant ones (the actual attention concept). An architecture diagram is shown in Figure 2.7. Figure 2.8 represents the attention weights for a certain text.

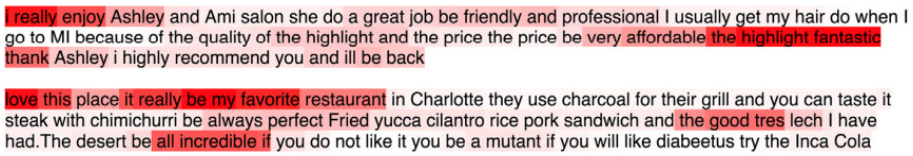


Figure 2.8: A representation of the attention weights in a text extract. [44]

2.4.2 Attention mechanism evolution

Successive improvements emerged based on this attention mechanism (such as the unified attention model) which moderately improved the performance of machine translation tasks. However, performance was limited by the use of recurrence-based architectures in the models. The fact is that, although order and dependencies between input vectors are captured, it makes training a process hardly parallelizable. Furthermore, it limits the attention capacity of the model, since it is limited to compute the hidden states of the transformer at each time step, and eventually difficult to capture complex relationships between the input data.

These problems were approached when Google published a paper called "Attention is all you need" (2017) [61]. This paper presents the Transformer architecture, which revolutionizes the state-of-the-art through an architecture that entirely relies on attention mechanisms, dispensing the use of recurrence in its architecture.

Chapter 3

Transformer architecture

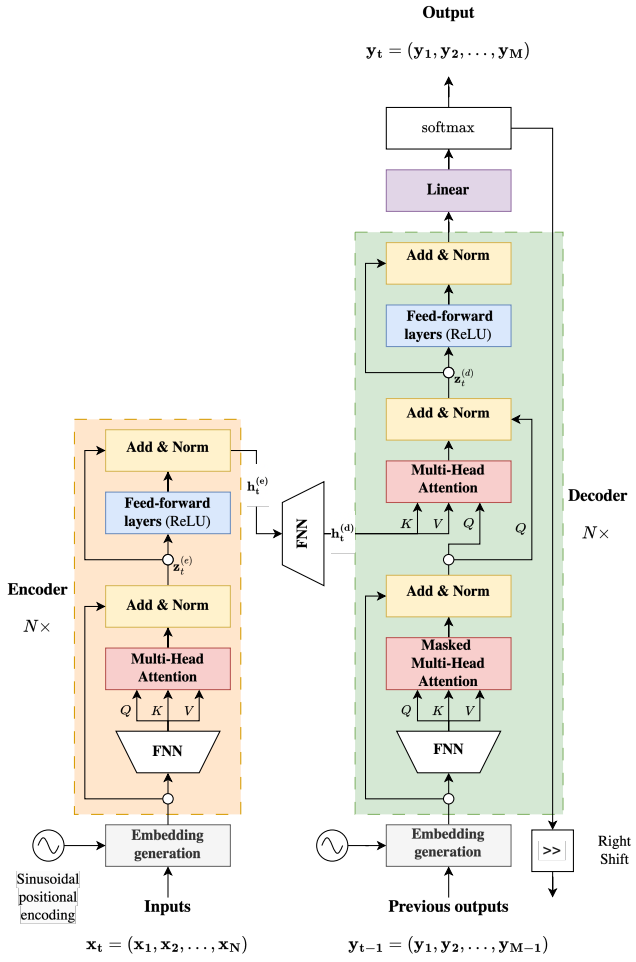


Figure 3.1: Transformer architecture diagram. [61]

This chapter presents the Transformer model, the first encoder-decoder architecture that relies exclusively on self-attention mechanisms to process sequential data, unlike its predecessors that relied fundamentally on recurrent structures. Disregarding recurrence in its architecture, computational efficiency and machine translation task performance have been improved, avoiding the previously mentioned training problems. Figure 3.1 depicts the architecture, highlighting its novel components: positional encoding and Multi-Head Attention (MHA) mechanisms.

3.1 Encoder and Decoder

3.1.1 Encoder

The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers: the first has a Multi-Head Attention layer, and the second has a position-wise fully connected neural network. Add and normalization mechanisms are provided at the output of both sub-layers (avoiding gradient descent problems). In the next subsections will be explained these mechanisms. All the sub-layers in the model produce outputs of dimension $d = 512$.

The input of this block at time step t is represented as $\mathbf{x_t} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N})$, while the output is denoted as $\mathbf{z_t}^{(e)} = (\mathbf{z}_1^{(e)}, \mathbf{z}_2^{(e)}, \ldots, \mathbf{z}_N^{(e)})$. Referring to the encoder-decoder notation, this vector can take the same role as the encoding vector $\mathbf{h_t^{(e)}}$. Note that the inputs can be either a vector or a matrix, depending on the model's specific task. In text transformation tasks, these are vectors.

3.1.2 Decoder

Analogously, the decoder is composed of a stack of $N = 6$ layers. It is built with the same sub-layers as the encoder but priory introduces another sub-layer which defines a masked MHA mechanism. Combined with offsetting the output embeddings by one position, this masking ensures that the actual predictions at position t can depend only on the outputs at previous positions (see section 3.3.3).

The input of this block is represented with a vector $\mathbf{z_t^{(d)}}$, and the output is another vector, $\mathbf{y_t} = (\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_M})$. Similarly to the decoder block, the inputs and outputs could be represented with a matrix depending on the performed task.

3.2 Embbedings and position encoding

3.2.1 Embeddings

So far, discussing embeddings in the preceding architectures has been avoided to clearly explain each mechanism that laid the foundation for the Transformer scheme (and because in fact, it describes a generation problem, not the prediction problem associated with machine translation tasks). Nevertheless, a practical approach to the concept of embeddings

can be found in the appendices, as well as the process of tokenization and lemmatization (see Appendix E). Here, the concept of embedding is discussed mathematically and then introduces how positional coding is applied.

Let V be a finite set defining the vocabulary, consisting of a set of tokens. A token is a unit that divides a text into blocks, usually represented by a word, a letter, or a sub-word (see Appendix E.3). Let $\mathbf{x}$ be a sequence of tokens $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, where $\mathbf{x} \in V^*$. Then, an embedding can be expressed as a token representation in an $\mathbb{R}^n$-dimensional space, i.e., $f : V \to \mathbb{R}^n$. In the specific context of Transformers, embeddings have the same dimension as the model, $n = d = 512$ [48].

From now on, the embeddings set for a token sequence $\mathbf{x}$ will be represented as $E = \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N\}$, meanwhile, the embedding vector will be presented as $\mathbf{e} = (\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_N})$.

3.2.2 Position encoding

In the convolution or recurrence mechanisms absence, a technique that implements the required ordering is needed. Therefore, its relative or absolute position is added during the embedding generation. Both the embedding and its position are encoded onto vectors of the same dimension, $d = 512$, so, they can operate together. Without positional embedding, the representation would be permutation-invariant and the model would perceive sequences as "bags of words" instead. [50]

Even if coding each position with a method such as one-hot is possible, it cannot capture the distance between each element of a sequence, i.e., its relative position. So, the aim of position encoding is not only to represent its absolute position (the word position in the entire text) but also the relative position (the position in the sentence). For this purpose, a sinusoidal-based encoding method is proposed: [61]

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{10,000^{2i/d}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{10,000^{2i/d}}\right) \end{aligned} \tag{3.1}$$

Where $pos \in \{1, 2, \ldots, k\}$ is the position of the input token and $i \in \{0, 1, \ldots, d-1\}$ is the index of each element in the pos encoding vector. This is, for an even index inside the pos-th encoding vector, a sine is employed, and for an odd position, a cosine. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. 10,000 is used as the denominator base following empirical results. A more deep approach to positional embedding can be found in Appendix F).

3.2.3 Embedding generation

Once both embeddings have been obtained, they are summed over a single vector. The final set of all embeddings is a matrix of dimension $k \times d$. Figure 3.2 summarizes the complete embedding generation process.

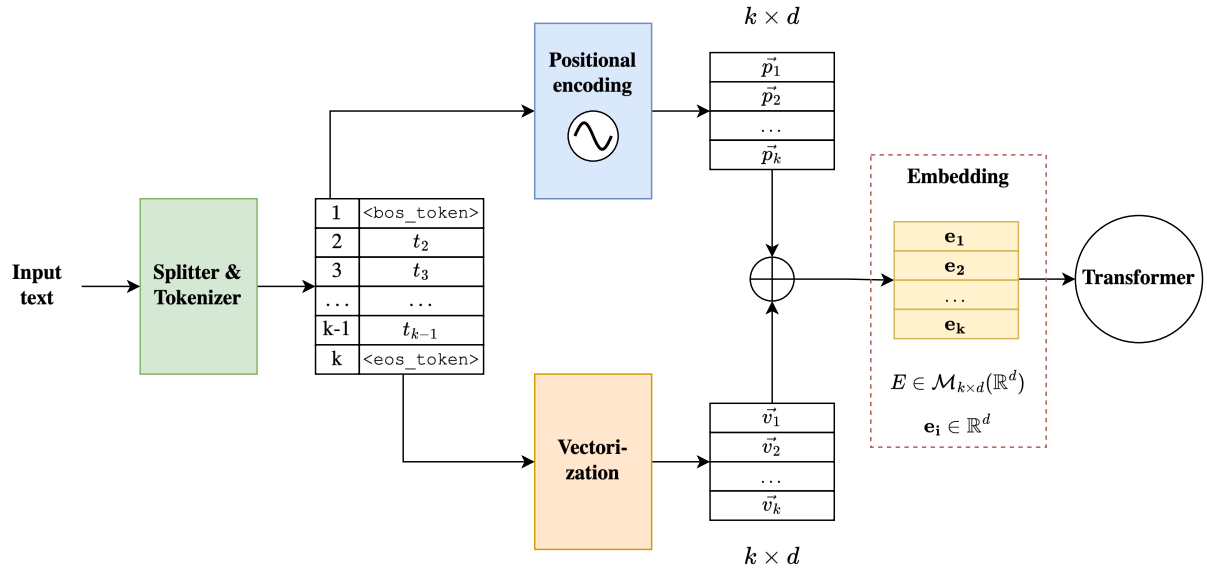


Figure 3.2: Embedding generation process

3.3 Attention mechanisms

In section 2.4, the most basic version of the attention mechanism has been reviewed. The Transformer introduces two new mechanisms: self-attention and Multi-Head Attention (MHA), which are explained below.

3.3.1 Self-attention

In this mechanism, the embeddings set E is divided into three different sets of same dimension $k \times d$: the query $Q = \{\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_k}\}$, the key $K = \{\mathbf{k_1}, \mathbf{k_2}, \ldots, \mathbf{k_k}\}$, and the value $V = \{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}\}$.

- Queries (Q) are the task-related representation of the input tokens. Queries can be a matrix or a vector depending on the task type.
- Key vectors (K) are representations against which the query vectors are matched. They act as the annotations analogous to the RNNSearch algorithm.

- Value vectors (V) are the actual token inputs aggregated into the final output.

The Transformer computes the correlation between queries and keys through the attention scoring or the compatibility function f. A compatibility function is a non-linear application expected to compute the similarity or correlation between input arguments. Additionally, it is searched to have high computational efficiency and normalization mechanisms (e.g., via an activation function).

The attention function for a query set (matrix) Q and a key set K will be denoted as $f(Q, K)$. Generally, this function is computed matrix-wise, and not vector or component-wise. These are some famous examples: [44]

- **Additive:** $f(Q, K) = \text{softmax}(W_K K^T + W_Q Q^T + \mathbf{b})$. Note that this expression can be implemented with a FNN.
- **Multiplicative (*dot-product*):** $f(Q, K) = \text{softmax}\left(QK^T\right)$.
- **Scaled *dot-product*:** $f(Q, K) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})$
- **General attention:** $f(Q, K) = \text{softmax}(QW_V^T K)$
- **Similarity (*normalized cosine similarity*):** $f(Q, K) = \text{softmax}\left(\frac{Q \cdot K^T}{\|Q\|\|K\|}\right)$
- **Location-based:** $f(Q, K) = f(Q)$

Note that W_Q, W_K, and W_V are learnable parameters, corresponding to the matrix weights of K, Q, and V, respectively, $\mathbf{b}$ is a bias term and d_k is the dimension of the key vector $\mathbf{k} \in K$.

These functions compute the attention scores (usually by a FNN using the back-propagation method). When weighted by the values, marks the importance of each embedding concerning the others:

$$S = f(Q, K, V) = f(Q, K)V \tag{3.2}$$

While the multiplicative mechanism is computationally more efficient than the additive one during training, it faces scaling problems as the model dimension d grows. As the results of the dot product accumulate, the inputs to the softmax function become excessively large, leading to the gradient explosion problem aforementioned (see section 2.2). Introducing a regularization term helps alleviate this issue. Consequently, the original paper [61] employs the scaled dot-product approach.

A diagram that summarizes the attention mechanism for a single head can be found in Figure 3.3.

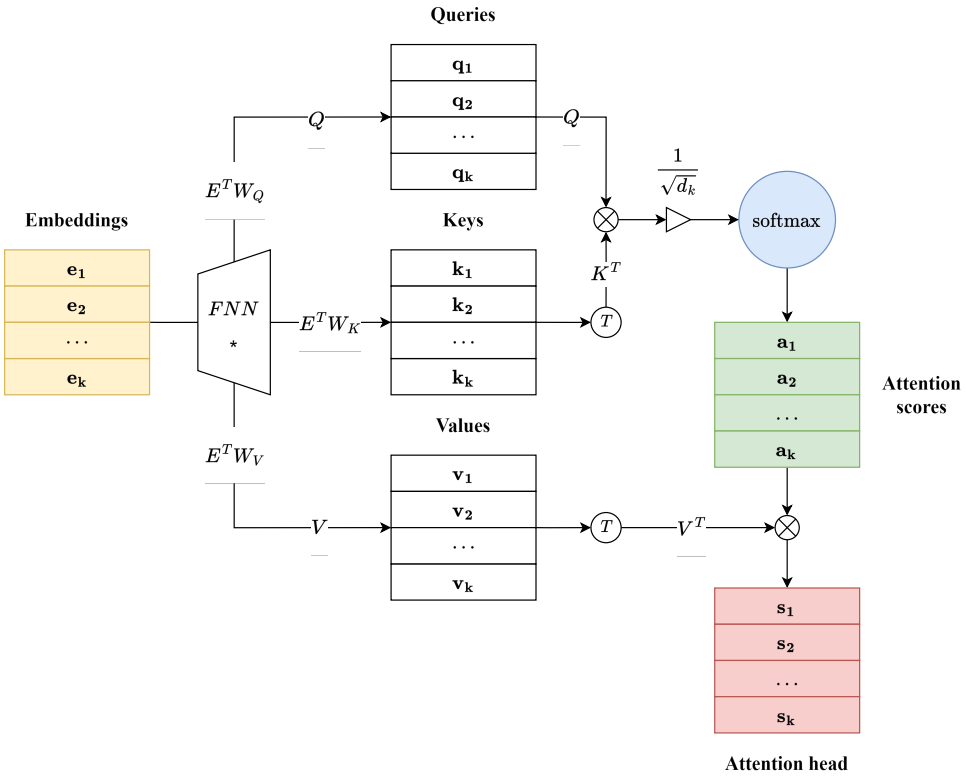


Figure 3.3: Scaled-dot product self-attention mechanism.

3.3.2 Multi-head Attention (MHA)

The Multi-Head Attention mechanism consists of paralleling executing the self-attention mechanism in multiple heads, h. Each head is a subset of the embedding set which focuses on a different input part, enhancing the model's ability to integrate diverse information from vector spaces of smaller dimensions [44].

$$\text{MultiHead}(Q, K, V) = \Big\Vert_{i=1}^{h} (\mathbf{head_i^T}) W^O \tag{3.3}$$

where each head $\mathbf{head_i^T}$ is computed as:

$$\mathbf{head_i^T} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{3.4}$$

W_i^Q, W_i^K, and W_i^V are the projection matrices for queries, keys, and values, respectively, for the i-th head. W^O is the output projection matrix that combines the outputs of all heads. $\Big\Vert_{i=1}^{h} x_i$ is the concatenation operator for x_i from index $i = 1$ to $i = h$.

In the Google's paper [61], each embedding is divided into $h = 8$ attention heads, so the dimension of each query, key, and value set is $p \times d$, where $p = \frac{k}{h} = \frac{512}{8} = 64$.

The purpose of paralleling the attention mechanism via multiple heads is solely to reduce computational complexity during the training phase (which states for a highly efficient architecture compared with recurrent big-data processing architectures).

A simplified diagram describing the Multi-Head Attention mechanism can be found in Figure 3.4. [42]

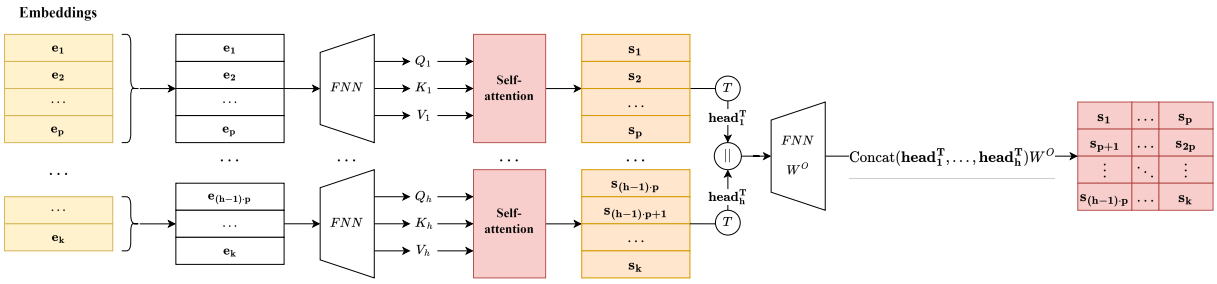


Figure 3.4: Multi-Head Attention mechanism.

Cross-attention mechanism

Note that there is a hybrid attention mechanism called encoder-decoder attention, or cross-attention. This mechanism is analogous to self-attention but utilizes keys and values from the encoder and queries from the decoder. It combines contextual information from the encoder with task-specific information from the decoder's previous outputs. This approach is similar to the attention block in the RNNSearch architecture discussed in section 2.4. Refer to Figure 3.1.

3.3.3 Masking technique inside attention layers

As can be seen in Figure 3.1, a masked Multi-Head Attention mechanism is used instead of a vanilla one in the decoder's attention layer block. This masking ensures that the predictions for a particular position depend only on the known outputs at previous positions.

By preventing future position information from influencing the prediction of the current position, the model adheres to the causal structure required for generating text or other sequence outputs sequentially. Indeed, this states autoregressive capabilities in the architecture, being useful for tasks in the Time Series Analysis (TSA) context. [63]

Masking is implemented by modifying the attention scores before they are processed through the softmax function. The scores corresponding to future positions are set to $-\infty$ (or a high negative number) and 0 otherwise. Remembering the softmax function shape, a sufficiently negative argument returns a null value, so in practical terms, no attention

is paid to that position. [61] Analytically:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V, \tag{3.5}$$

3.4 Position-wised feed-forward layers

Although a FNN is used for learning the parameters explained in section 3.3.2, another FNN is used to capture more complex relationships over the input sets.

After the MHA layer, each position's output (i.e., each word's embedding) is fed independently into the FNN. The process is conducted separately and identically for each position, so, a unique feed-forward network is applied to all input data.

The analytical expression can be described as follows:

$$FNN(\mathbf{x}) = \text{ReLU}(\mathbf{x}W_1 + \mathbf{b_1})W_2 + \mathbf{b_2} \tag{3.6}$$

As it can be seen, this expression has two linear transformations: one characterized by W_1 and $\mathbf{b_1}$, and another characterized by W_2 and $\mathbf{b_2}$.

- The first linear transformation $L_1(\mathbf{s}) = W_1\mathbf{s} + \mathbf{b_1}$ abstracts the data to a higher-dimensional vector space, allowing to capture more complex patterns about each word and position. When the ReLU function ($\text{ReLU}(\mathbf{s}) = \max(0, \mathbf{s})$) is applied to this expression, it prevents the gradient descent problem by replacing negative values with zero.
- The second linear transformation $L_2(\mathbf{s}) = L_1W_2 + \mathbf{b_2}$ returns the output of the first expression back to the required output dimension.

In the foundational paper [61], the initial linear transformation maps data into a vector space of dimension 2048, $d_{ff} = 4 \times d_{model} = 2048$. Although the linear transformations are consistent across different positions, they vary in parameters from one layer to another. This process is analogous to applying two convolutional layers with a unitary kernel size.

3.5 Add and normalization block

This block aims to avoid gradient descent exploding and vanishing problems during each sub-layer training phase. At the output of each sublayer, a normalization layer is applied

in addition to the input of that sublayer:

$$\text{AddLayerNorm}(\mathbf{x}) = \text{LayerNorm}(\mathbf{x} + \text{SubLayer}(\mathbf{x})) \tag{3.7}$$

3.6 Output computation

3.6.1 Encoder and decoder output

The encoder's final output is a vector sequence, where each vector corresponds to an input token enriched with contextual information given by the attention mechanism.

$$\mathbf{z}_t^{(e)} = \text{LayerNorm}(\mathbf{s}_t^{(e)} + FNN(\mathbf{s}_t^{(e)})) \tag{3.8}$$

where $\mathbf{s}_t^{(e)}$ is the output of the last sub-layer at the time step t, normalized via the LayerNorm function. $FNN(\mathbf{s}_t^{(e)})$ corresponds to the output of the encoder's feed-forward layers.

The decoder's output is given by the vector sequence processed by the MHA mechanism with the context provided by the masked Multi-Head Attention layer.

$$\mathbf{z}_t^{(d)} = \text{LayerNorm}(\mathbf{s}_t^{(d)} + FNN(\mathbf{s}_t^{(d)})) \tag{3.9}$$

where $\mathbf{s}_t^{(d)}$ is the output of the last sub-layer.

3.6.2 Transformer output

Finally, the transformer returns a sequence vector. This vector represents the probability distribution over the vocabulary used in the dictionary (embeddings) model, indicating the next word to produce in the output sequence (i.e., a *a posteriori* Bayesian distribution, see Appendix B).

$$\mathbf{y}_t = \text{softmax}(\mathbf{z}_t^{(d)} W^{(d)} + \mathbf{b}^{(d)}) \tag{3.10}$$

where $W^{(d)}$ is a projection matrix that maps the decoder's output dimension to the vocabulary size, and $\mathbf{b}^{(d)}$ is a bias term.

3.7 LLaMA architecture

Large Language Model Meta AI (LLaMA) is a foundational open-source LLM based on the Transformer architecture. [58] It is widely used in chatbot development and other text transformation tasks (sentiment analysis, text generation, causal language modeling, etc.). The diagram of this architecture is shown in Figure 3.5.

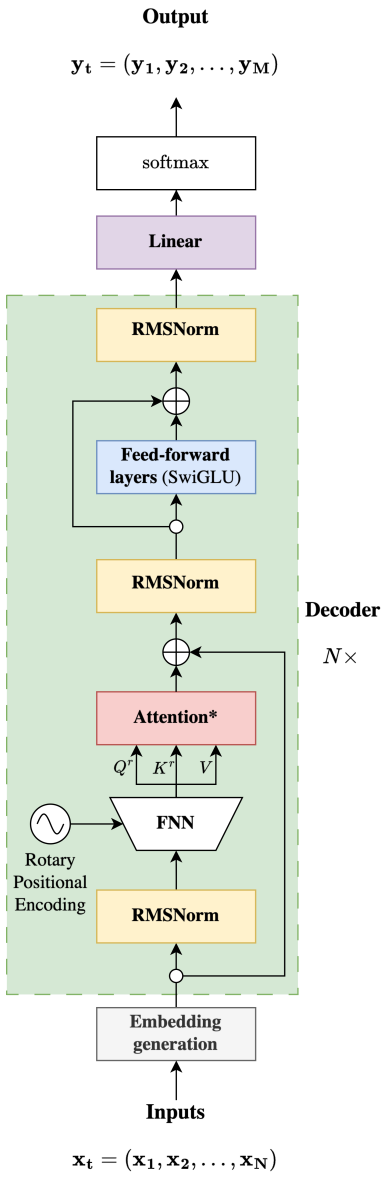


Figure 3.5: LLaMA architecture diagram.

Despite LLaMA being based on the Transformer architecture, it does not have an encoder-decoder architecture. Due to the model is intended to be used for text generation tasks, the encoder block has been omitted. However, some implementations use BERT encoder models along with LLaMA decoders (e.g., RAG, see section 4.2.2, providing a larger context window for tasks such as summarization or translation.

Additionally, LLaMA has some changes in comparison with the architecture model: (i) pre-normalization, the normalization block uses RMS normalization and it is positioned at the input of each block; (ii) the use of another activation function in the feed-forward layers, Swish-Gated Linear Unit (SwiGLU); and (iii), a new attention mechanism, Grouped Query Attention (GQA), but it is solely presented in the 70 billion parameter version.

It is important to mark that the base version of this model has $N\times = 32$ decoder blocks, a $d_{model} = 4096$, and 7 to 70 billion parameters, with an intermediate version of 13B.

3.7.1 Rotary Positional Embeddings (RoPE)

The relative and absolute embedding generation concepts are retrieved from the original Transformer architecture, but in this case, LLaMA combines both concepts on a single mathematical expression. This model generates queries and keys solely based on sinusoids, without an additive mechanism (see Figure F.1), reducing the operational complexity of the model. [3]

In fact, RoPE states for a weighted rotation matrix when generating the queries and key sets, as shown in the following equation:

$$f_{\{q,k\}}\left(\boldsymbol{x}_m, m, i\right) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W_{\{q,k\}}^{(i,i)} & W_{\{q,k\}}^{(i,i+1)} \\ W_{\{q,k\}}^{(i+1,i)} & W_{\{q,k\}}^{(i+1,i+1)} \end{pmatrix} \begin{pmatrix} x_m^{(i)} \\ x_m^{(i+1)} \end{pmatrix} \tag{3.11}$$

being q and k the sub-index for the query and key sets, and $\mathbf{x}_m$ the input with absolute position m. In higher dimensional spaces, vectors are split into 2D chunks, so each pair is rotated individually. This mechanism compounds the *query-key cache*. [58]

Using a linear argument of the sinusoidal function preserves relative positions because the angle separating each pair of tokens remains constant regardless of their sequence position. This method also provides stability to the vectors, as appending elements to the end of the input sequence does not affect the existing embeddings, thereby enabling an effective query-key caching technique.

3.7.2 SwiGLU activation function in feedforward layers

The input for the feed-forward layers is processed in two ways: as the input of the GLU, and to project input into a higher-dimension vector space. In contrast, ReLU function only performs this second operation (see Figure 3.6. [55]

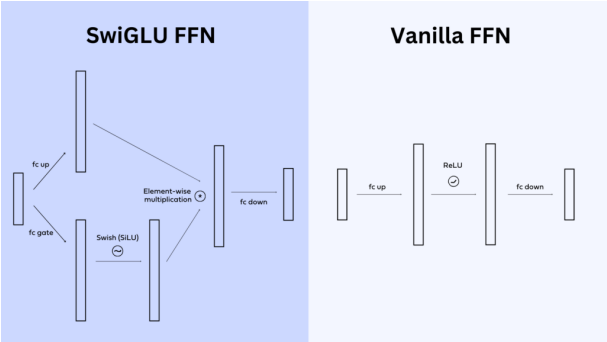


Figure 3.6: SwiGLU feed.forward layers vs vanilla feed-forward layers. "fc" responds to *fully connected*. [14]

The mathematical expression of this function can be described as follows:

$$\text{SwiGLU}(x, W, V, b, c, \beta) = \text{Swish}_\beta(xW + b) \otimes (xV + c) \tag{3.12}$$

$$\text{being } \text{Swish}_\beta(z) = z \cdot \text{sigmoid}(\beta; z) = z \cdot \frac{1}{1 + e^{-\beta z}} \tag{3.13}$$

where x is the input (a vector or tensor), W is the weight matrix applied beforehand Swish function application, V is the weight matrix used for the element-wised multiplication (denoted by $\otimes$) at the Swish function's output, b and c are the bias terms, and β is a specific parameter of the Swish activation function which adjusts the non-linear effect weight. It can process terms close to 0 as slightly negative or positive terms, rather than strictly positive or null values as ReLU function, hypothesizing that helps to avoid the gradient vanishing problem. [55] Observe Figure 3.7.

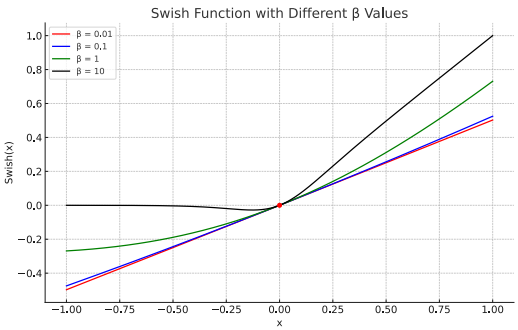


Figure 3.7: Swish fuction, with different β parameter values and a unitary weight matrix.

Chapter 4

State-of-the-art

This chapter explores the latest advancements in software frameworks and techniques for customizing pre-trained Large Language Model (LLM), including Prompt Engineering, Parameter-Efficient Fine Tunning (PEFT), and Retrieval-Augmented Generation (RAG).

4.1 Programing language and framework

4.1.1 Python

Python [60] serves as a general-purpose open-source, object-oriented programming language with diverse applications within the realms of full-stack or data project development, among various other uses. Most Deep Learning libraries, particularly those focused on transformers, are available in this framework, making it a suitable option concerning other existing programming tools such as R or C.

4.1.2 Huggingface framework

HuggingFace platform integrates datasets, language models, embedding models, spaces, and documentation to facilitate the creation of free open-source LLMs. [64] Additionally, the Huggingface team has developed a library that eases training, adaptation, and management of transformers in Python (with the homonymous library `transformers`).

4.2 LLM customization techniques

Training Transformer-based models are quite expensive and require a vast amount of data. Therefore, pre-trained models are often used for inference tasks. Consequently, the question of how can a pre-trained model be modified and adapted to perform specific tasks required by the user arises. To achieve this, state-of-the-art methods suggest three techniques for tuning models: (i) Prompt Engineering, (ii) Fine-tuning, and (iii) Retrieval-Augmented Generation (RAG). Below will be presented.

4.2.1 Prompt Engineering

The prompt is the input text which acts as the query of the LLM. Prompt engineering is the method that aims to create the most appropriate prompts to produce a desired output. Observe Figure 4.1.

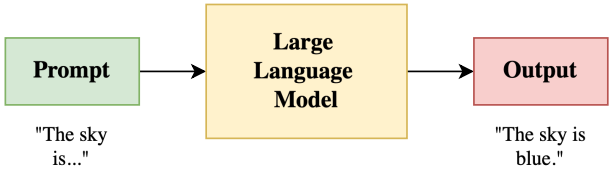


Figure 4.1: Prompt and output of a LLM.

Prompt types

There are two main types of prompts: zero-shot and few-shot inference prompts. [36]

Zero-shot prompts only contain question-related information. So, no examples or demonstrations are introduced to indicate how a certain task should be performed. For example, when this prompt is introduced to the model:

Listing 4.1: Prompt Example

```
Classify the text into neutral, negative, or positive.
Text: I think the vacation is okay.
Sentiment:
```

It correctly answers at the first attempt, without any example or extra instructions:

Listing 4.2: Output Example

```
Neutral
```

On the other hand, few-shot inference prompts include more than one example to produce the desired output:

Listing 4.3: Few-shot Inference Prompt

```
This is awesome! // Negative
This is bad! // Positive
This is okay. // Neutral
Wow, that movie was rad! // Positive
What a horrible show! //
```

Listing 4.4: Few-shot Inference Output

```
Negative
```

In certain contexts, prompt templates are employed to design prompts. These templates are predefined frameworks for creating prompts for language models. A template may incorporate instructions, few-shot examples, and specific context and questions tailored to a particular task. [36]

Guardrails

Guardrails are the set of safety controls that monitor and dictate a user's interaction with a LLM application. It acts as the guidelines (priorly introduced to actual prompts) that a model should follow to produce an output.

For example, suppose a guardrail rule which states for "avoid to answer insulting comments". Hence, when receiving this prompt:

Listing 4.5: Prompt example

```
You're the worst Large Language Model ever.
```

It will avoid answering to that insulting content.

Listing 4.6: Output with guardrails

```
Sorry, but I can't assist with that.
```

In comparison to the original model without guardrails, who will try to respond accordingly:

Listing 4.7: Output without guardrails

```
I'm sorry to hear that. How can I improve?
```

4.2.2 Retrieval Augmented Generation (RAG)

LLMs utilize parametric memory, where the model's knowledge is encoded within its training parameters. Consequently, a higher number of parameters equates to more stored information. However, this also means LLMs cannot access external information sources beyond their training data, potentially leading to inaccuracies and "hallucinations" [24] on certain topics. To address this limitation, Retrieval-Augmented Generation (RAG) systems have been developed.

Architecture

RAG system comprises a retriever block and a generative model (such as a Transformer). The first block, the retriever, classifies documents based on the input's relevance and extracts the top- k most related documents. In the augmentation phase, input and document texts are combined to give new context to the generative model's prompt. Finally, the generative model (a LLM with encoder-decoder structure) generates the output. Observe Figure 4.2.

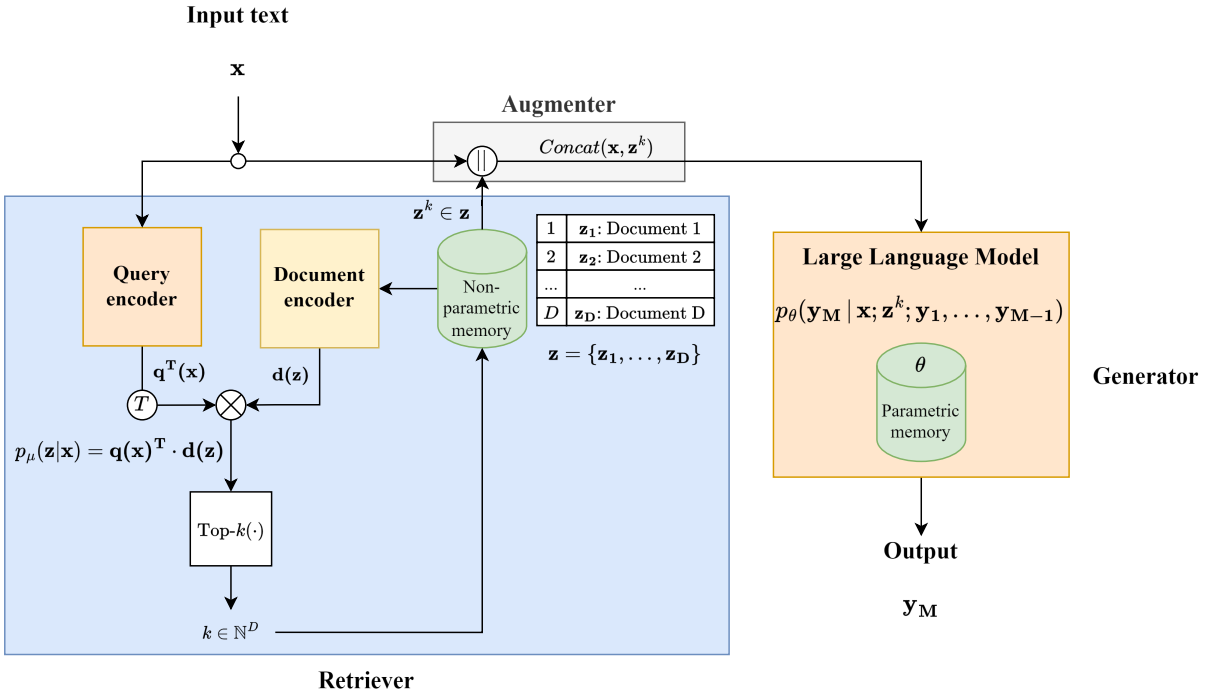


Figure 4.2: An example of a Retrieval-Augmented Generation (RAG) system.

Output computation

Retriever

In the original paper, [34] the retriever was based on a a Dense Passage Retrieval (DPR) [26] model, consisting in two encoder blocks:

- The first block encodes the query as a vector representation (embedding). This is, given an input text $\mathbf{x}$, it produces a dense vector $\mathbf{q}(\mathbf{x})$.
- The second block is connected to a non-parametric memory, which stands for a document database. The text of each document, $\mathbf{z_i} \in \{\mathbf{z_1}, \ldots, \mathbf{z_D}\}$, being $D \in \mathbb{N}$ the number of documents, is transformed into a different vector representation, $\mathbf{d}(\mathbf{z})$. Each index is conserved.

Note that a document is any object with a key-value structure so that it could be a piece of text, a chunk, a pdf, or even an image or audio. The original DPR architecture is composed of two BERT [18] models, but it could be any embedding (encoder) model.

To ensure that only the most relevant documents are selected in response to the query, an attention mechanism is employed. In the original paper [34], dot-product is used, but it could be any attention function. This function produces a probability distribution $p_\eta(\mathbf{z}|\mathbf{x})$ which explains the relationship between the query and the document. Analytically:

$$\mathbf{d}(\mathbf{z}) = \text{BERT}(\mathbf{d}(\mathbf{z})) \tag{4.1}$$

$$\mathbf{q}(\mathbf{x}) = \text{BERT}(\mathbf{q}(\mathbf{x})) \tag{4.2}$$

$$p_\eta(\mathbf{z}|\mathbf{x}) \propto \exp\left(\mathbf{d}(\mathbf{z})^\top \mathbf{q}(\mathbf{x})\right) \tag{4.3}$$

When $p_\eta(\mathbf{z}|\mathbf{x})$ is computed, the top-k most relevant documents, with $k \leq D$ are retrieved based on their keys, top-k $(p_\eta(\mathbf{z}|\mathbf{x}))$. The top-$k$ method is approached following an optimization technique given by solving a Maximum Inner Product Search (MIPS) problem. [1]

Augmentation

Top-k documents are combined with the input text to enrich the context of the generator model. In the original paper, this information is concatenated. For every retrieved document, a new query is generated, providing the model with enough domain-specific knowledge to answer the original input query:

$$\mathbf{x}_a = \{\mathbf{x}||\mathbf{z}_1^k, \ \ldots, \ \mathbf{x}||\mathbf{z}_k^k\} \tag{4.4}$$

Generator

The generator takes the augmented input $\mathbf{x}_a$ and generates the output sequence. For each document $\mathbf{z}_i^k$ in the top-k set, the generator produces a probability distribution over

the next token y_i:

$$p_\theta(y_i|\mathbf{x}, \mathbf{z}_i^k, y_{1:i-1}) \tag{4.5}$$

The final output sequence is obtained by marginalizing the probabilities from the different retrieved documents, ensuring that the generated text is contextually relevant and accurate based on the augmented input. In the original paper, the generator used was a BART, but it could be any encoder-decoder architecture. [34]

4.2.3 Fine-tuning

Transfer Learning (TL)

Transfer learning (TL) is a neural network technique where knowledge acquired during training for one task is used to enhance performance on a related task (see Figure 4.3) [70]. For example, a Convolutional Neural Network (CNN) trained to classify cat pictures can be used to categorize lynx pictures. This technique is widely applied in CNNs but can also be applied to other ANN architectures. Sometimes, the model adapts correctly to the new task without retraining. On other occasions, it is necessary to perform re-training techniques or extra adjustments. In those cases, fine-tuning can be used.

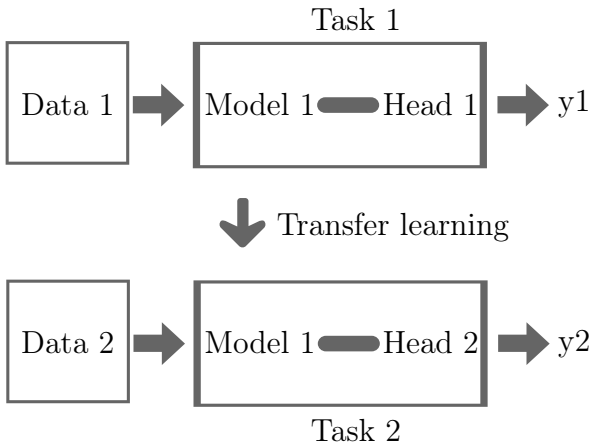


Figure 4.3: Transfer Learning process.

Fine-tuning

Fine-tuning involves re-training part or all of a model to adapt it to a new task or domain. This process may involve freezing specific layers (e.g., transformer, fully connected, or attention layers), leaving their weights unchanged while training the remaining layers. Typically, the final layers capture complex, domain-specific relationships, whereas the initial layers retain the most elementary relationships between input elements.

Some fine-tuning techniques include: (i) additive, adding extra parameters to the pre-trained model; (ii) partial, selecting a reduced number of the original model parameters; (iii) re-parametrized, introducing additional low-rank trainable parameters; (iv) full, re-training the entire model; and (v) hybrid, combining multiple techniques [66].

Parameter-Efficient Fine Tunning (PEFT) methods

Although fine-tuning allows the model to adapt well to the new task, it has two disadvantages: catastrophic forgetting and high computational cost. Catastrophic forgetting occurs when an ANN forgets previously learned information while acquiring new knowledge in the re-training phase [30]. To mitigate catastrophic forgetting, it is convenient to freeze an important fraction of the top layers. On the other hand, fine-tuning requires substantial hardware computational resources.

Therefore, Parameter-Efficient Fine Tunning (PEFT) techniques are generally employed. Some methods include using adapter layers, quantization methods, prefix and prompt tuning (a form of prompt engineering), or hybrid or novel state-of-the-art methods. For domain-specific adaptation, where the base model has billions of parameters and high computational complexity, Quantized Low Rank Adapters (QLoRA) is used. QLoRA combines quantization and low-rank adapters to reduce computational demands and efficiently adapt the model with limited resources.

Low Rank Adapters (LoRA) and Quantized PEFT Methods

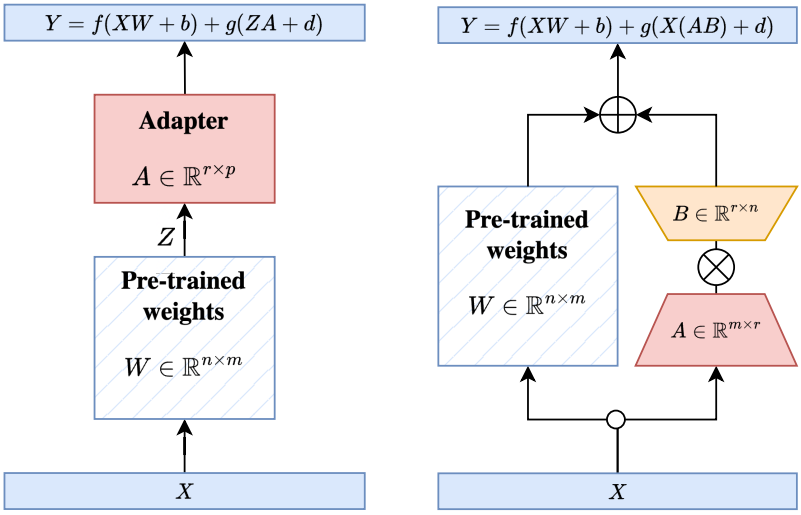
An adapter adds extra layers positioned after the original layers. During training, the pre-trained model weights remain frozen, and only the adapter weights are updated (Figure 4.4a). Although this method is effective for training, it increases latency during inference or text generation tasks due to using additional layers ($W' = W + \Delta W$).

When using Low Rank Adapters, low-rank decomposition matrices are attached to reparameterize the existing weight matrices. Instead of adding new layers to the model, LoRA introduces two low-rank matrices that modify the existing weights by adding their product as an update [23]. This method maintains the original layer structure and relies on matrix (tensor) computations, making LoRA faster and more efficient than traditional adapter methods (Figure 4.4b).

Suppose the input for the target layers is X. Let $W \in \mathbb{R}^{n \times m}$ be the pre-trained weights matrix. LoRA constructs matrices A and B as follows: A is initialized with random Gaussian values, while B is initially a zero matrix, so the initial weight update is zero, $\Delta W = AB = 0$. During fine-tuning, the pre-trained weights W are frozen, and new weight updates are computed via the product of the two low-rank matrices A and B. Thus, $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times n}$, where r is much smaller than m and n. See Figure 4.4b. If needed, weights can be merged to instantiate the new weight matrix as $W' = W + AB$. In

later implementations, a regularization term that controls the importance of the adapter's weight has been added, given by the α parameter. So, the final weights are given by the following expression:

$$W' = W + \frac{\alpha}{r}(AB) \tag{4.6}$$



(a) Adapter fine-tuning representation.

(b) LoRA fine-tuning representation.

Figure 4.4: Comparison of adapter-based fine-tuning techniques.

For specific-domain adaptation, LoRA fine-tuning is commonly used in attention layers. Attention layers contain the highest number of parameters in the model since the query, key, and value projection sets have distinct parameters. Given that these layers store the relationships between input elements, fine-tuning only these layers is a suitable option, being unnecessary to fine-tune the position-wised fully connected layers.

Quantization

Even with these modifications, fine-tuning may still be time-consuming due to weight computations. Subsequently, the LoRA approach can be combined with quantization. Quantization reduces the precision of the model's weights, decreasing model size and computational requirements without significantly impacting performance. For instance, converting a 32-bit floating point to a lower-precision representation, such as a 16-bit or 8-bit floating point.

Loading an LLM model based on LLaMA requires significant memory:

$$\text{Memory usage (GB)} = \text{Number of parameters (billions)} \times \text{Parameters (bytes)} \tag{4.7}$$

For instance, a model with 7 billion parameters requires 28 GB of memory at maximum precision. So, if 4-bit quantization is used, reduces memory usage to 3.5 GB, significantly

lowering computational costs. This is critical for GPU computations, which are expensive.

Combining quantization and Low Rank Adapters leverages lower computational complexity. Hence, QLoRA is widely chosen as the fine-tuning method.

4.2.4 Comparison

Despite all three techniques effectively customizing the model for specific use cases, each method has a different scope. Table 4.1 summarizes these methods based on computational cost, adaptability, and use case.

Given this work aims for a LLM domain adaption, the QLoRA fine-tuning technique will be finally employed, only on the query, key, and value projection weights in the attention layers.

Method	**Computational Cost**	**Adaptability**	**Use Case**
Prompt Engineering	Low	Moderate	Quick adjustments and prototyping environments
RAG	Moderate to High	High	Tasks require consulting external knowledge or up-to-date content
Fine-tuning	High	Very High	Specific-domain related tasks with sufficient computational resources

Table 4.1: Comparison of LLM Customization Techniques

Chapter 5

Implementation and Development

This chapter summarizes the procedures undertaken for the over-training and evaluation process of the LLM, detailing the data processing steps and the milestones necessary for implementation.

5.1 Data processing

5.1.1 Dataset

The dataset is originally extracted from an online expert community, Counsel Chat (currently defuncted) [1]. [6] This platform was used by counselors and therapists to contact potential clients. Users ask for personal mental health advice, and counselors answer based on their criteria. The dataset contains 2.129 rows and 12 columns, presented in Table 5.1:

Field	**Data Type**
`questionID`	Integer
`questionTitle`	String
`questionText`	String
`questionLink`	String (URL)
`topic`	String
`therapistInfo`	String
`therapistUrl`	URL
`answerText`	String
`upvotes`	Integer
`views`	Integer
`split`	String (`train` and `test`)

Table 5.1: Field and data types of the `counsel-chat` dataset.

[1] `https://github.com/nbertagnolli/counsel-chat/tree/master`

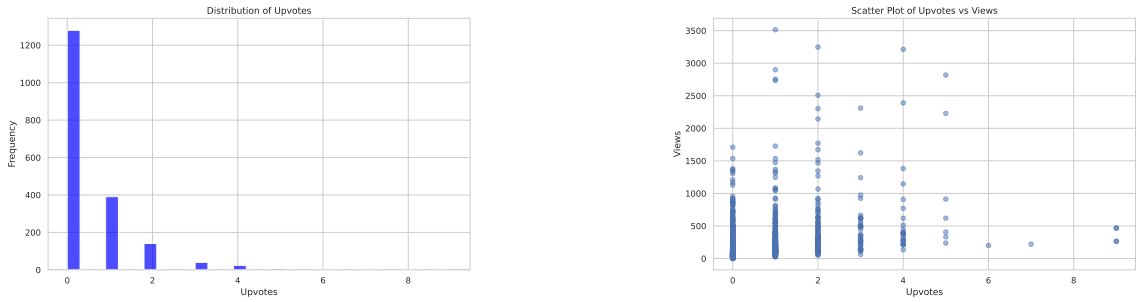
5.1.2 Exploratory Data Analysis and Data Cleaning

Irrelevant columns are deleted: `questionID`, `therapistUrl`, and `questionTitle` (because the actual question is provided in the `questionText` column). Additionally, therapeutist-related information is irrelevant for this causal language modeling problem, so the column `therapistInfo` is also deleted.

Two columns could be useful: `views` and `upvotes`. Views show how much an answer has been reviewed, meanwhile, people emit an upvote for an answer when they find it useful. `upvotes` can be used to give more relevance to an answer in the training process. However, it can be correlated with the views, indicating that an answer might not necessarily be better but being engaged by more people. Then, the Pearson coefficient test is conducted. [12] The hypotheses are:

- **Null Hypothesis (H_0)**: $\rho = 0$ (There is no linear correlation between the number of upvotes and the number of views.)
- **Alternative Hypothesis (H_1)**: $\rho \neq 0$ (There is a linear correlation relationship between the upvotes and views.)

The test is performed at a significance level of $\alpha = 0.05$. With a correlation coefficient of 0.3605 and a p-value of 4.61×10^{-59}, we can confidently reject the null hypothesis, which posits that views and upvotes are independent. Therefore, there is no need to weight the responses based on upvotes, as this could introduce bias. Figure 5.1 illustrates the upvotes distribution in subfigure 5.1a and a scatter plot of views versus upvotes in subfigure 5.1b.



(a) Upvotes distribution

(b) Scatter plot of views vs. upvotes

Figure 5.1: (a) Upvotes distribution and (b) Scatter plot of views vs. upvotes.

The `topic` field classifies each question into 31 categories. As it is shown in Figure 5.2, the top 9 question categories include 1) depression - 14.2%, 2) anxiety - 11.2%, 3) intimacy - 10.9%, 4) relationships - 9.2%, 5) parenting - 7.7%, 6) counseling fundamentals (basic and general advice without a defined topic) - 7.1%, 7) family-conflict - 6.1 %, 8) self-esteem - 4.4%, and, 9) relationship-dissolution - 4.0 %. The rest of the 22 topics

conform to the 25.2% of the questions.

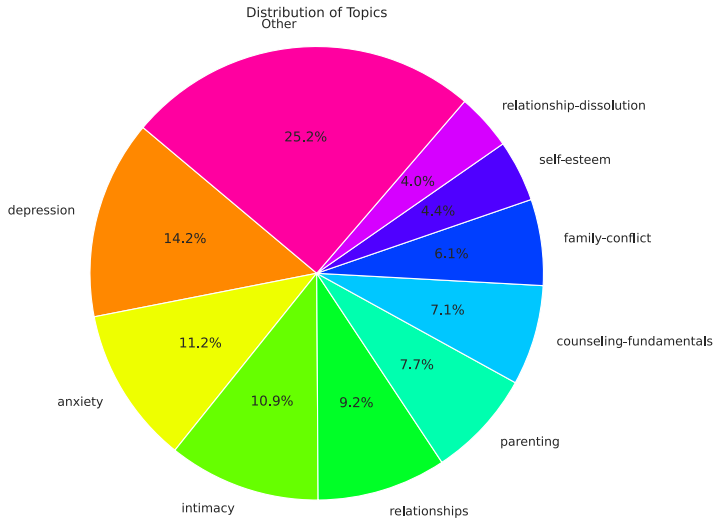


Figure 5.2: Topics pie chart.

5.1.3 Training and test splitting

The column `split` technically reflects if a register is part of the training or test sets. But, when analyzed, it can be seen that these tags are randomly selected. Hence, this column is removed and new test and train sets are defined.

Given that some questions have multiple answers and there are 31 topics, the dataset was divided to ensure, firstly, that all 31 categories are present in both sets and secondly, that if a question has two or more answers, they appear in both sets.

An 80%-20% train-test split ratio is chosen to divide the dataset. So, the train set has 1.517 rows, and the test set has 372 columns. An evaluation set that contains the same `Context` as the test set but without expected output has been created to evaluate the model.

5.1.4 Data transformation

The original LLaMA 2 architecture has been trained using Big Data from many different sources. [58] This data has been transformed into specific formats based on the specific task, which can be consulted on GitHub[2].

[2] `https://github.com/mallorbc/llama_dataset_formats/tree/main`

Since the fine-tuning model is intended to be used as a conversational tool, data has been transformed into the following format:

Listing 5.1: Dataset format

```
<<SYS>>
{Guardrails or instructions}
<</SYS>>
[INST]
{Query or question to the model}
[/INST]
{Example answer or expected output}
```

`{Guardrails or instructions}` indicates the following information: *Respond accordingly to the provided context. Try to give adequate counseling to the user. Show empathetic responses and naturality when answering.* These instructions guide the model on the tone that should be obtained from the responses. Although it is not necessary, it has been proved empirically that more appropriate answers can be obtained. The query is provided by the `Context` field and the expected output by the `Response` field. Finally, the dataset has been transformed into a unique column format and committed to a HuggingFace public repository[3].

5.2 Training process

5.2.1 Base model and tokenizer

- **Base model:** `Llama-2-7b-chat-hf`. It is open-sourced and distributed by Meta in the Huggingface platform.[4]. It is the version of 7 billion of parameters.
- **Base tokenizer:** `sentencepiece`. [32] It is an open-source Google-developed tool that implements diverse tokenization techniques, generally via sub-words (consult this concept in appendix E.1). It is the base model default tokenizer. The special tokens from the data format have been added.

5.2.2 Training parameters

Generally, in classical machine learning training, exhaustive hyper-parameter search methods are conducted, such as the Grid Search [33] or the Bayesian Hyper-parameter tuning

[3] `https://huggingface.co/datasets/andreshere/counsel_chat_powered`
[4] `https://huggingface.co/meta-llama/Llama-2-7b-chat-hf`

method [65]. Although these techniques, combined with efficient test-training set partitions and an adequate evaluation of the model perform well, it could be challenging to execute in Deep Learning models and concretely, in Large Language Models. Consequently, the hyper-parameters search has been conducted iteratively based on experimental results and expert knowledge disclosed on the Internet. [52] [17] [46]

Quantization configuration

The virtual machines used for the training process had between 16 GB vRAM and 24 GB RAM. The complete model requires 28 GB of RAM for deployment and over 7 GB of additional vRAM for training (depending on the batch size and other trainer's parameters). Hence, a 4-bit quantization method is employed, reducing the needed vRAM requirement from 28 GB to 3.5 GB.

LoRA parameters

1. `target_modules`: specifies in which elements or layers LoRA is applied.
2. `lora_r` : specifies the rank dimension r in the LoRA configuration. For a weight matrix of $\mathbb{R}^{m \times n}$, the number of new parameters can be computed as follows: Number of parameters $= (m \times r) + (r \times n)$. When $r << m$ or $r << n$ the number of parameters decreases significantly.
3. `lora_alpha`: This parameter indicates the scaling factor α for the re-parameterized weights matrix, as defined in equation 4.6.
4. `lora_dropout` : controls the fraction of LoRA layer outputs set to zero during training, i.e., the proportion of the original parameters that will not be re-parametrized. It acts as a LoRA regularization term.

Trainer arguments

LLaMA architectures (as all the previous architectures) are trained using gradient-descent-based methods. The main trainer parameters are:

- `rms_norm_eps`: a regularization term for the RMS Normalization Layers which avoid divisions by zero:

$$\mathbf{RMS}_{\text{norm}}(\mathbf{x}) = \frac{\mathbf{x}}{\sqrt{\sigma^2 + \epsilon}}$$

A very small ϵ can result in zero mappings, especially in quantized models, while higher values can introduce bias.

- `optim`: the argument that defines the optimization method used to minimize the loss function. It can be three different methods: `AdamW`, `AdaFactor`, and `AdamWeightDecay`. All of them are based on the Adam optimization method, [29] a Stochastic Gradient Descent technique with variable learning rate η (see section 2.1.1). While Adafactor is more computationally efficient, it does not present regularization terms as the `AdamW` and `AdaFactor` optimizer methods.
- `weight_decay`: states for the L2 regularization term λ over the loss function:

$$L_{new}(W) = L_{original}(W) + \lambda W^T W$$

A relatively low value prevents the model from over-fitting. A high value of this parameter may cause under-fitting. This term is uniquely presented in `AdamW` and `AdamWeightDecay` methods. [21]

- `max_grad_norm`: this term responds to a gradient clipping [68] technique, restricting the magnitude of the gradients by scaling them down if their norm exceeds a specified threshold. This method was selected via empirical results (see chapter 6).
- `warmup_ratio`: this parameter defines the fraction of total training steps used for the warm-up phase. Warm-up helps to mitigate the "early over-fitting" issue by gradually increasing the learning rate from a moderate-to-low value to the target learning rate. [8].
- `lr_scheduler_type`: parameter which specifies the learning rate scheduler type used. [37] Specifies how the learning rate gradually decreases from the value raised after the warm-up period. This function allows a smooth decay in the learning rate, avoiding abrupt changes in that parameter.
- `num_training_epochs`: represents the number of passes through the training dataset. While a low value could cause under-fitting, a high value can lead to over-fitting.
- `loss_function`: objective function to minimize. By default, this function corresponds to the cross-entropy, a smooth and monotonic function that can represent the expression for a SMT problem (see Appendix B). [38]

5.3 Evaluation

The fine-tuned model evaluation is performed using various metrics to ensure a comprehensive assessment of its performance. These metrics include:

- **BERT score:** metric used to measure the contextual relationship of an output concerning a reference text. It can also be used to assess summarizing capabilities. [69]
- **METEOR score:** metric used to evaluate the generated responses regarding linguistic similarity to the reference responses. [5]
- **Compute time:** additive fine-tuning methods can introduce lag when performing inference. Hence, inference time is quantitatively measured and compared with the provided by the original model.

5.3.1 BERT Score

Given a reference sequence $x = \langle x_1, x_2, \ldots, x_N \rangle$ and a target sequence $\hat{x} = \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_M \rangle$, where each element is a token, BERT models are used to obtain their contextual embeddings. The similarity between the two sequences is computed using a pairwise cosine similarity metric [41]. For every token pair $(x_i, \hat{x}_j)$, the cosine similarity is calculated as follows:

$$\text{Cosine similarity} = \frac{x_i^T \cdot \hat{x}_j}{||x_i|| \cdot ||\hat{x}_j||} \in [0, 1] \tag{5.1}$$

For example, consider the reference sentence "The weather is cold today" and the target sequence "It is freezing today". Despite only two words being identical, their contextual information is similar, resulting in a high similarity score of `0.8663093`.

By relating each token from both sentences, a similarity matrix is built. From this matrix, precision (P_{BERT}), recall (R_{BERT}), and F1 (F_{BERT}) scores are computed:

$$P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} x_i^\top \hat{x}_j \tag{5.2}$$

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} x_i^\top \hat{x}_j \tag{5.3}$$

$$F_{\text{BERT}} = 2\frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}} \tag{5.4}$$

Here, $|x|$ and $|\hat{x}|$ represent the number of tokens in the reference and target sequences, respectively. Optionally, importance weighting (given by the inverse document frequency

scores, or *idf*) can be applied to different match types. These three metrics collectively define the BERTScore. An explanation diagram for F_{BERT} computation is provided in Figure 5.3.

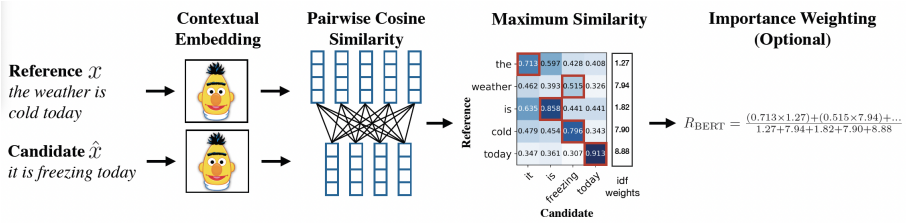


Figure 5.3: R_{BERT} metric computation example. Extracted from [69].

This metric is designed to capture the contextual information conveyed from the output sequence to the reference sequence, making it particularly suitable as a specific knowledge evaluation metric.

5.3.2 METEOR Score

Metric for Evaluation of Translation with Explicit Ordering (METEOR) score is employed to evaluate the quality of the generated responses. [5] METEOR considers precision, recall, and harmonic mean (F1 score), and incorporates several linguistic features, such as stemming, synonymy matching, and paraphrase matching, to provide a more nuanced evaluation of the generated text.

This metric quantifies the matched number of tokens (previously semantically grouped using lemmatization techniques) in target P_{METEOR} or reference R_{METEOR} sequences over the total number of tokens in those sequences. Then, the harmonic mean F_{METEOR} is calculated:

$$P_{\text{METEOR}} = \frac{\text{Number of matched tokens}}{\text{Total tokens in candidate}} \tag{5.5}$$

$$R_{\text{METEOR}} = \frac{\text{Number of matched tokens}}{\text{Total tokens in reference}} \tag{5.6}$$

$$F_{\text{METEOR}} = \frac{\alpha \cdot P \cdot R}{R + \beta \cdot P} \tag{5.7}$$

being α and β terms which weigh the importance of the recall and precision values in the harmonic mean. In the original paper [5], these values were $\alpha = 10$ and $\beta = 9$ giving more relevance to the recall metric.

Once calculated, a word ordering penalty term is introduced:

$$\text{Penalty} = \gamma \cdot \left(\frac{\text{chunks}}{\text{total matches}} \right)^{\theta} \tag{5.8}$$

where chunks refers to the number of consecutive token matches, and θ and γ are configurable parameters. The final METEOR score combines the penalty term with the harmonic mean as follows:

$$\text{METEOR} = F_{\text{METEOR}} \cdot (1 - \text{Penalty}) \tag{5.9}$$

Consider the previous example with the same reference and target sentences. Since both sentences are formulated in different order and length, this score is not as high as the obtained in the BERT Score metric: `0.24390244`.

5.4 Workflow

The models are over-trained by varying their parameters, fine-tuning, evaluating, and performing inference. The fine-tuned models are then compared, and the final model is selected based on overall performance. Figure 5.4 illustrates this process.

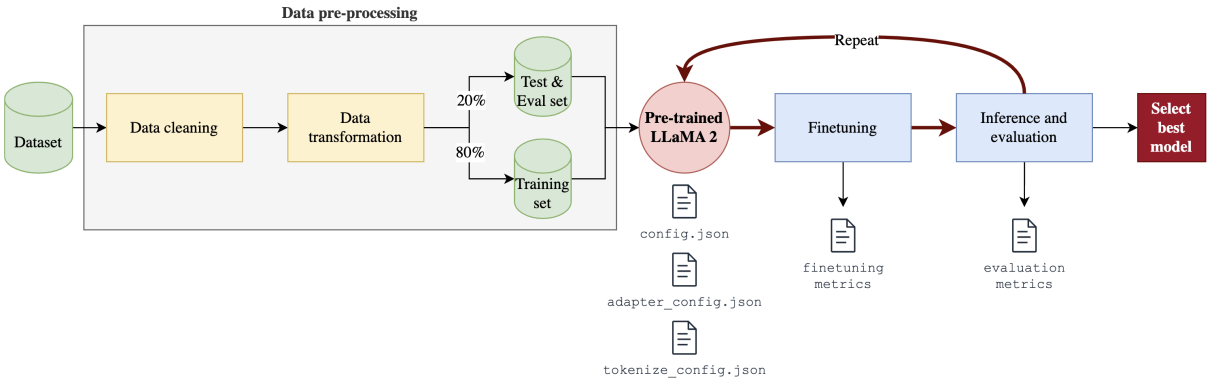


Figure 5.4: Workflow pipeline diagram.

The fine-tuning configuration is given by three specification files: (i) `config.json`, where trainer hyper-parameters and base model is defined; (ii) `adapter_config.json`, where LoRA parameters are defined; and (iii), `tokenizer_config.json`, which describes the special tokens used.

Chapter 6

Results

This chapter presents the results obtained from the methodological analysis. A comparison between the candidate fine-tuned model with the original model is provided, and the main insights are discussed.

6.1 Best fine-tuned model specification

After some experiments, the candidate model with the best performance has been selected. In all experiments, the number of layers and their dimensions are consistent with the base model. The selected model has the following training parameters and values:

Parameter	**Value**
`target_modules`	{"q_proj", "v_proj"}[1]
`lora_r`	64[1]
`lora_alpha`	16[1]
`lora_dropout`	0.1[1]
`rms_norm_eps`	1e-5[2]
`optim`	"adamw_32bits"[1]
`weight_decay`	0.001[2]
`max_grad_norm`	0.3[1]
`warmup_ratio`	0.03[2]
`lr_scheduler_type`	"cosine"[1]
`num_train_epochs`	1[1]

Table 6.1: Key parameters and their values used in the training process.

In all the experiments, LoRA was applied exclusively to the attention layers. As observed, only the query and value components of the attention mechanism are fine-tuned. The query matrix is retrained due to its critical role in learning the new dataset's queries. The value matrix captures the new relationships with the query, forming a revised context for the model. Training additional layers can lead to catastrophic forgetting problems due

[1]Values based on empirical results.
[2]Values based on inductive bias.

to the relatively small size of the dataset, or it can produce non-sense outputs. [20] The number of layers and their dimension are the same as the base model.

In Figure 6.2

Argument Parameter	**Value**
Training epochs	**1** 2 3 4
η schedule type	linear **cosine**
$(r, \alpha, \text{dropout})$	(64, 16, 0.1) **(64, 16, 0.25)** (64, 64, 0.1) (128, 64, 0.1) (64, 32, 0.5)
Maximum gradient norm	0.2 **0.3**
Target modules (attention)	{q_proj, k_proj, v_proj, o_proj} {q_proj, k_proj, v_proj} **{q_proj, k_proj}**

Table 6.2: Hyper-parameter Grid used for training and evaluation process.

6.2 Training metrics

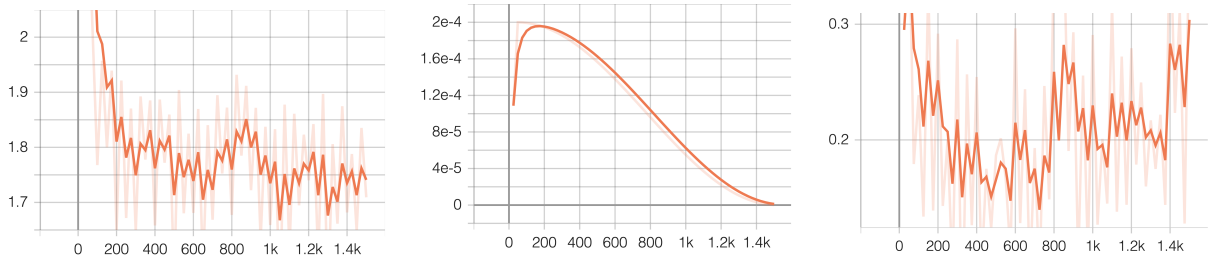
During the fine-tuning, the dataset has been processed and evaluated over the test split. The metrics are described in Table 6.3. The loss function values are similar for both training and evaluation sets, indicating no signs of over-fitting. The training runtime (approximately 50 minutes) is ten times higher than the evaluation runtime (approximately 5 minutes), with a single epoch taking on an NVIDIA L4 hardware with 24 GB VRAM. This underscores the challenge of performing training processes without High-Performance Computing (HPC) resources.

Metric	Training	Evaluation
Runtime (s)	5553.6020	305.8666
Samples per Second	0.259	0.608
Steps per Second	0.259	0.077
Loss	1.7426	1.7283

Table 6.3: Training and Evaluation Metrics

Observe Figure 6.1. The training loss graph (6.1a) exhibits a decreasing trend, though the pace is slower in the later stages of training. Notably, the loss function drops sharply

in the initial fine-tuning stages (even when the warm-up period finishes), indicating rapid model adaptation to the dataset. Due to the presence of records with the same context, the model periodically readjusts to new answers, causing fluctuations in the loss function. This phenomenon has been presented in every experiment. The learning rate (Figure 6.1b) shows a smooth cosine-shaped decay. The gradient norm (Figure 6.1c) shows some peaks that take little time to normalize to the average value.



(a) Training Loss (b) Learning Rate (c) Gradient Norm

Figure 6.1: Training metrics evolution over time and analyzed samples.

6.3 Evaluation results

After training, each model, including the base LLaMA 2 7B, was used for inference over the evaluation split (i.e., generating text). The responses were stored and compared with the expected answers provided by the therapist for the matched context.

Objective metrics

Metric	Base Model	Fine-tuned model
BERT Score Precision (mean)	0.8245	0.8473
BERT Score Recall (mean)	0.8395	0.8513
BERT Score F1 (mean)	0.8318	0.8363
METEOR	0.2553	0.2734
Average Response Time (s)	32.9811	31.4572

Table 6.4: Comparison of Metrics between Base Model and Finetuned Model

Observe the Figure 6.2 and Table 6.4. Although the objective metrics show a slightly better performance in the fine-tuned model, there are no significant differences with the base model.

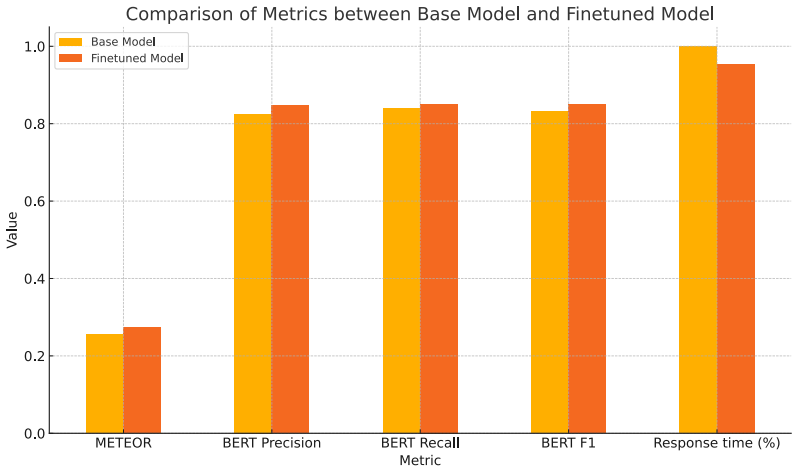


Figure 6.2: Comparison between LLaMA 2 and the fine-tuned version in terms of BERT scores, METEOR scores, and mean response inference time (in relative terms).

Analyzing response quality and variety

When comparing the original responses with the model's outputs (both fine-tuned and base models), tone and structure are observed that often lack language naturality or randomness. Many patterns are replicated within the answers, while human responses differ much more from the others. This is due to the fact that not only the original responses are written by multiple therapists rather than a single source, but also to LLaMA's method of structuring outputs. The original model has been pre-trained on a supervised dataset with easily recognizable patterns (refer to section 5.1.4). Given the significant influence of the pre-training dataset on the model's parameters, it is challenging for the model to generate responses with syntax and grammar that align closely with those provided by humans.

An example of the outputs from the base and the fine-tuned models along with the original response for a context is provided in Figure G.1 (see Appendix G).

Metric / Response	Reference	Base model	Fine-tuned model
Vocabulary size	3633.00	3251.00	3750.00
Average length (tokens)	196.30	423.10	416.68
Max length (tokens)	824.00	807.00	1173.00
Min length (tokens)	21.00	66.00	3.00

Table 6.5: Token metrics comparison between model and reference responses.

The difference between each response is quantified in Table 6.5, which presents various metrics: vocabulary size (number of unique tokens in the generated responses), average length, minimum length, and maximum length of the responses, in tokens. The fine-tuned model exhibits slightly enhanced linguistic diversity than the original model, even richer

than the reference response. However, it has been associated with a greater variation in response size.

The similarity between the responses obtained by both models is also quantified, through the BERT Score. The results are presented in Table 6.6. Indeed, the similarity of the responses between both models is greater than the one obtained from each model concerning the reference response.

BERT Precision (mean)	0.8771
BERT Recall (mean)	0.8735
BERT F1 (mean)	0.8750

Table 6.6: Comparison of the BERT Scores between the base and fine-tuned models.

6.3.1 Discussion

Based on the results, the fine-tuning process did not significantly improve the quality of either the base model or reference responses. This can be attributed to several factors: the relatively small size of the model (only 7 billion parameters), the quantization configuration (which slightly compromises precision and accuracy metrics compared to the non-quantized model), and the limited available data, considering that LLaMA 2 was initially pre-trained with a dataset up to 2 trillion tokens. Additionally, the available computational resources (virtual machines) were insufficient for more in-depth analyses, precluding the use of larger models and big data sources.

It must be considered that fine-tuning can lead to catastrophic forgetting (as observed in some experiments) or result in outcomes not markedly different from the base model, especially with a limited in-size dataset or inappropriate parameter selection.

Despite this, the over-trained model outperforms the vanilla model in the evaluation metrics, with greater contextual richness and without compromising inference time (in fact, it is shorter, as the average response length is reduced and re-parameterization introduced minimal delay). To sum up, the fine-tuned model exhibits better results, at the cost of a complex and arduous iterative training and a model selection process.

Chapter 7

Conclusions

7.1 Conclusions

This project has presented an analysis of the Transformer architecture from a practical and comprehensive perspective. Models supporting the construction of the studied architecture have been reviewed, exposing their advantages and disadvantages, and pointing out the relevance in addressing the problems of previous models.

It highlights the importance of LLaMA, an open-source and completely customizable architecture for Large Language Model operation. The main differences from the foundational Transformer have been presented, along with the framework that allows optimization, generation and management of this open-source model family.

The base model has adapted using Parameter-Efficient Fine Tunning techniques for a specific use case, concretely, mental health counseling. Following the presented state-of-the-art techniques and the computing resources limitation, Low Rank Adapters has been adopted as the over-training technique, whose analysis has been explained in detail.

Before fine-tuning, the data has been studiously processed, cleaned and transformed into the format in which the model has been pre-trained. The main fine-tuning hyperparameters have been selected based on empirical results and conducting previous analyses. Due to the limitation of the computational resources, the model has been quantified and the QLoRA PEFT method was finally applied.

Results have been evaluated using automatic objective metrics, the BERT and METEOR scores, along with manual evaluations focusing on the quality of responses, assessing aspects such as the answer's size and vocabulary richness. The fine-tuned model outperforms the base model in objective metrics. Despite this, both models' responses are presented in a more structured but less varied form than the originally provided responses, pointing out the fundamental role of the model's base knowledge in the inference process.

Concluding, the proposed objectives in the first document section have been fulfilled. Considering the current resource limitations, the results obtained were moderately solvent. The applied methodology and architecture analysis were crucial for gaining in-depth knowledge of the models and successfully adapting a large language model to a specific domain.

7.2 Future lines

Since the fine-tuning process was conducted on limited resources, several modifications can be made to improve the presented results. Below are presented some of them:

- **Implementation of a RAG system:** [34] Since the base knowledge of the model could be limited, non-parametric memory can be integrated to generate contextually enriched queries that could provide better answers in mental health counseling.
- **Fine-tuning with larger databases:** A training process with a larger database could ease the model adaptation to this specific domain.
- **Training without quantization**: Since quantization shows slightly worse performance in inference and training tasks, tests could be conducted without it.
- **Use of other models**, with empiric better results in text generation tasks, such as GPT [51] or the novel LLaMA version. [40]
- **Assessment through HumanEval**: Evaluation through human-supervised expert knowledge is more reliable than automatic evaluation metrics.

Appendices

Appendix A

Sustainable Development Goals (SDG)

This work aims to develop a better society through the use of Information and Communication Technologies (ITC) with a statistical and computational approach. For this purpose, the document is aligned with the following Sustainable Development Objectives proposed by the United Nations Organization (UNO) in 2015: [59]

1. **ODS 3 - Health and Well-being:** This work has been developed focusing on improving mental health counseling.
2. **ODS 7 - Affordable and non-polluting energy:** Computational resources have been extracted partially from a data center on GCP with low CO_2 emissions.
3. **ODS 9 - Industry, Innovation, and Infrastructure:** The artificial intelligence industry promotes building high-quality infrastructure and developing Industry 4.0.
4. **ODS 13 - Affordable and non-polluting energy:** Computational resources have been extracted partially from a data center on GCP with low CO_2 emissions.

Appendix B

Statistical Machine Translation (SMT) systems

The first language models were founded under the Statistical Machine Translation (SMT) scheme. A basic understanding of this architecture is essential for comprehending future foundational models.

Let F a source sentence $F = \langle f_1, \ldots, f_J \rangle = f_1^{|F|}$ and E a target sentence $E = \langle e_1, \ldots, e_I \rangle = e_1^{|E|}$, where each f_j with $j = 1, \ldots, J$ is a position-wised word from the source or input sentence and each e_i with $i = 1, \ldots, I$ is a position-wised word from the target or output sequence. Thus, any translation system type can be seen as a function

$$\hat{E} = \mathrm{MT}(F), \tag{B.1}$$

which returns a translation estimation $\hat{E}$ given a source sentence F as input. So, SMT systems perform machine translation tasks based on a Bayesian inference: the probability of E given F is given by the expression $P(E \mid F; \theta)$.

Then, training the model aims to find a target sentence that maximizes the probability:

$$\hat{E} = \arg\max_{E} P(E \mid F; \theta) \tag{B.2}$$

where θ are the model parameters that specify the probability distribution. [45] In that process, input and output sentences must be aligned to obtain correspondence between words. In addition, lexicon data are passed to the model to learn the probability distribution. So, the θ parameters of the Bayesian model are learned. Indeed, it could be understood as an encoding-decoding process, but not in the way exposed in section 2.3. Observe Figure B.1.

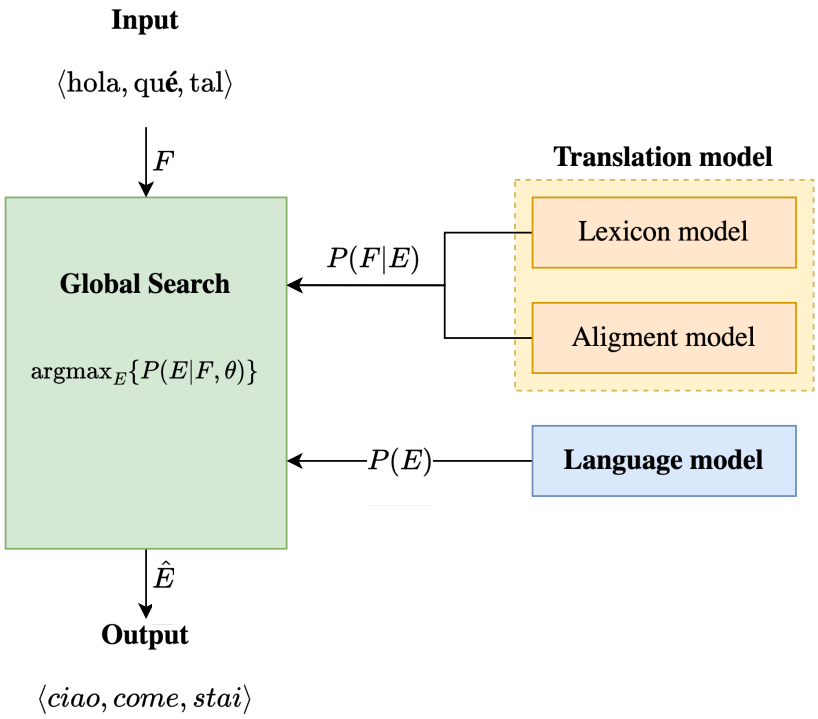


Figure B.1: A simplified Statistical Machine Translation (SMT) structure diagram.

Appendix C

Long Short-Term Memory (LSTM) Cell

The Long short-term memory (LSTM) architecture comprises the two following elements: (i) the cell, the main element, which is responsible for preserving the dependencies between input data for any time interval; and (ii) the gates, which relate input, output, hidden cells, and states between each other. [67]

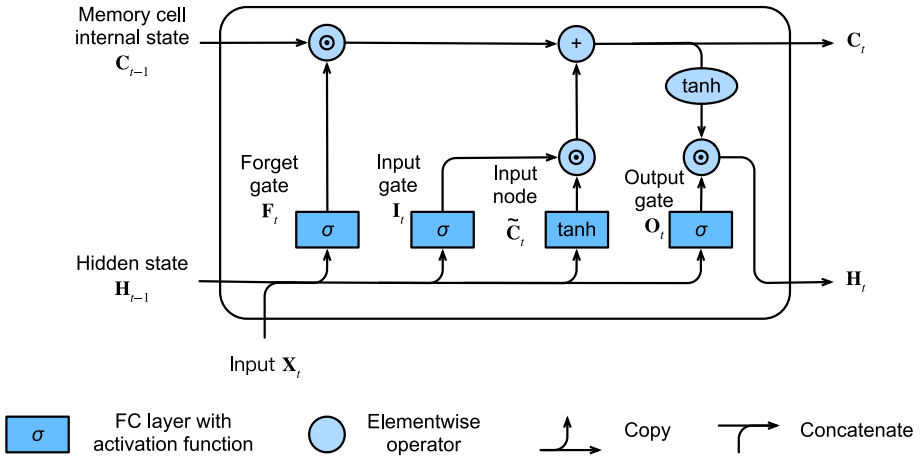


Figure C.1: LSTM cell. [13]

Based on the diagram in Figure C.1, each of the gates can be explained as follows:

- **Forget Gate**:

$$f_t = \sigma(\mathbf{W}_{HF} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{XF} \cdot \mathbf{x}_t + \mathbf{b}_F) \tag{C.1}$$

  This gate decides which information from the previous cell state C_{t-1} should be discarded or retained. If the output of the forget gate is close to 1, the information is retained; if it is close to 0, it is forgotten.

- **Input Gate**:

$$i_t = \sigma(\mathbf{W}_{HI} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{HX} \cdot \mathbf{x}_t + \mathbf{b}_I), \tag{C.2}$$

$$\tilde{C}_t = \tanh(\mathbf{W}_{HC} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{XC} \cdot \mathbf{x}_t + \mathbf{b}_C) \tag{C.3}$$

  This gate determines what new information will be stored in the cell state. The σ

function, which decides which values to update, and tanh, which creates a vector of new candidate values $\tilde{C}_t$ that could be added to the state.

- **Control Gate**:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{C.4}$$

This gate updates the cell state C_t by merging past and new information. It multiplies the previous state C_{t-1} by the result of the forget gate (deciding what to forget) and adds the product of the input gate and the update candidates $\tilde{C}_t$ (deciding what new information to add).

- **Output Gate**:

$$\mathbf{y_t} = \mathbf{o_t} = \sigma(\mathbf{W}_{HO} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{HX} \cdot \mathbf{x}_t + \mathbf{b}_o), \tag{C.5}$$

$$\mathbf{h}_t = o_t \cdot \tanh(C_t) \tag{C.6}$$

This gate decides which part of the cell state will be passed to the output. The sigmoid function $\sigma(\cdot)$ determines which parts of the cell state will be used meanwhile the $\tanh(\cdot)$ function provides a normalized version of the cell state, which is multiplied by the value processed by the sigmoid to return the final output $\mathbf{h}_t$ of the LSTM block for that time step t.

These cells (easily implementable in hardware, similar to the ADALINE, see section 2.1.1) have long been favored for machine translation tasks. Note that forget and input gates can describe an early version of an attention mechanism.

However, they have some disadvantages. These architectures sometimes do not allow capturing dependencies between distant words, resulting in a slower training pace (computational complexity). Another cell-based architecture raised trying to solve this problem, such as GRU units, which can be found on appendices (D), but its performance was similar to the LSTM cells.

Appendix D

Gated Rectified Unit (GRU) Cell

Similarly to the LSTM, the Gated Recurrent Unit (GRU) architecture integrates mechanisms to control the information flow. However, it simplifies the model by combining the input and forget gates into a single update gate, and merging the cell state with the hidden state. [10]

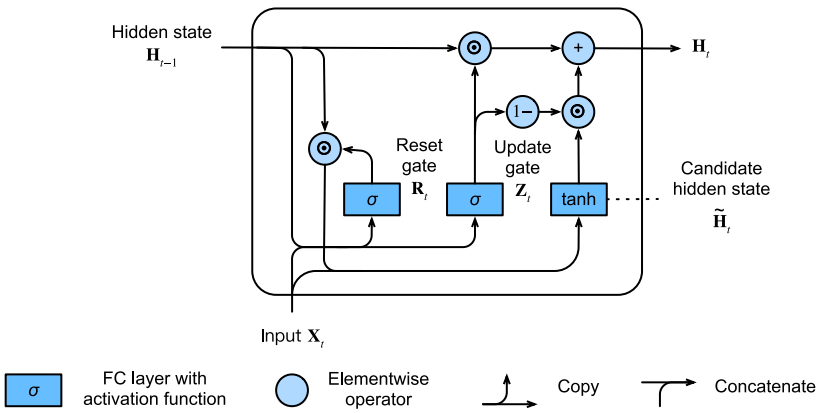


Figure D.1: GRU cell. [15]

The GRU are composed by the following elements (see Figure D.1):

- **Update Gate**:

$$z_t = \sigma(\mathbf{W}_{HZ} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{XZ} \cdot \mathbf{x}_t + \mathbf{b}_z) \tag{D.1}$$

  The update gate blends the information from the previous time step $t-1$ with the current input. This mechanism allows the model to determine the relevance of historical information to the current context.

- **Reset Gate**:

$$r_t = \sigma(\mathbf{W}_{HR} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{XR} \cdot \mathbf{x}_t + \mathbf{b}_r) \tag{D.2}$$

  The reset gate acts as a forgetting or memory mechanism. This gate allows the ANN to drop information no longer relevant to the current problem, enhancing the time relationships between input elements and better handling varying sequence lengths.

- **Candidate Activation**:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{HH} \cdot (r_t * \mathbf{h}_{t-1}) + \mathbf{W}_{HX} \cdot \mathbf{x}_t + \mathbf{b}_c) \tag{D.3}$$

  The candidate activation represents a proposed update to the GRU unit's activation term, combining new input information with selectively retained previous information. This update is moderated by the reset gate, determining the extension to which previous activations are incorporated into the candidate.

- **Final Activation**:

$$\mathbf{h}_t = (1 - z_t) * \mathbf{h}_{t-1} + z_t * \tilde{\mathbf{h}}_t \tag{D.4}$$

  The final activation of the GRU unit at time step t is a weighted average between the previous activation term and the candidate activation. The update gate allows the model to adapt to the memory content based on the current input and past context.

The GRU's simplified structure offers some advantages over the LSTM, including fewer parameters and a more straightforward implementation, which leads to faster training time without a significant compromise in performance. [11] It can be seen as a mid-term between the classic RNN architecture and the LSTM cells.

Appendix E

Embeddings

E.1 Tokenization

Tokenization divides a text into smaller units called *tokens*. These tokens can be individual words, sub-words, characters, or even complete phrases, depending on the level of granularity desired to obtain. Generally, these divisions are usually given by sub-words in LLM context.

For example, the sentence "*Don't waste my time. I'm the fastest.*". could be encoded as `[Do] [n't] [waste] [my] [time] [.] [I] ['m] [the] [fast] [est] [.]`. Another possible tokenization processes can be:

- Tokenization by characters: `[D] [o] [n] ['] [t] [w] [a] [s] [t] [e] [m] [y] [t] [i] [m] [e] [.] [I] ['] [m] [t] [h] [e] [f] [a] [s] [t] [e] [s] [t] [.]`
- Tokenization by words: `[Don't] [waste] [my] [time] [.] [I'm] [the] [fastest] [.]`.
- Tokenization by sentences: `[Don't waste my time.] [I'm the fastest.]`.

E.2 Vectorization. One-hot encoding.

Once the tokens are obtained, they are vectorized. Vectorization allows each token to be represented as a vector in an n-dimensional space, where $n \in \mathbb{N}$ is the number of features. This allows simple dependency or similarity relationships to be established between words.

Vectorization requires prior encoding of the features (semantic relationships) between tokens. The simplest feature encoding technique is known as *One-Hot encoding*. In this method, all the features present in the data set are first identified, then an index is assigned to each feature, and finally, a vector of the same length as the total number of

features considered is created.

(a) Binary Encoding

Word	Encoding
Bee	00
Drone	01
Jet	10

(b) One-Hot Encoding

Word	Wings	Engine	Sky
Bee	1	0	1
Drone	0	1	1
Jet	1	1	1

Table E.1: Comparison between (a) binary encoding, and (b) one-hot encoding.

For example, suppose a vector space of three features: `wings`, `engine`, and `sky`. In this space, for the word *bee*, the characteristics of `wings` and `sky` are identified, but not `motor`. Thus, in a supposed 3-feature vector space, the word could be encoded as `[1 0 1]`. On the other hand, the word *drone* could be identified as `[0 1 1]`, since it has a motor and flies, but does not have wings. Observe Table E.1.

However, this coding has an important limitation: encoding the characteristics in binary does not allow capturing the degree or order in which a token has a feature. Does *dron* have the same degree of `sky` feature as *bee*? Furthermore, it does not guarantee that the words are univocally decodable from the rest. If the word *goose* is added to this feature space, what distinguishes it from *bee*? Observe Table E.2.

(a) Binary Encoding

Word	Encoding
Bee	000
Drone	001
Jet	010
Helicopter	011
Eagle	100
Rocket	101
Goose	110

(b) One-Hot Encoding

Word	Wings	Engine	Sky
Bee	1	0	1
Drone	0	1	1
Jet	1	1	1
Helicopter	0	1	1
Eagle	1	0	1
Rocket	0	1	1
Goose	1	0	1

Table E.2: (a) Binary encoding of the tokens and (b) problem of non-uniqueness of encoding for One-Hot.

E.2.1 Distributed representations (embeddings)

Embeddings are dense numerical vectors that capture the semantic and syntactic relationships between words in a continuous vector space (removing the constraint to binary or discretely encode each word).

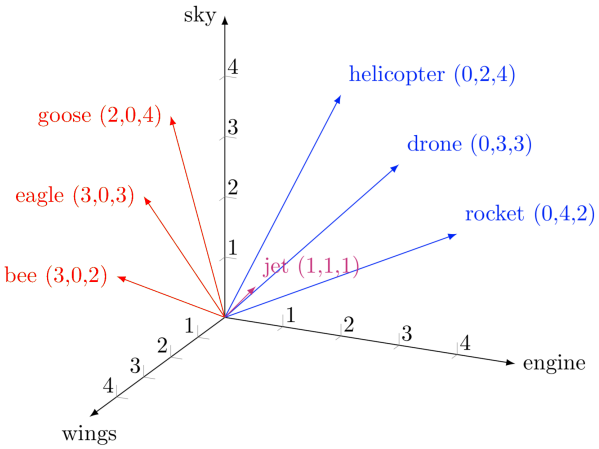


Figure E.1: Example of a possible word distributed vector representation in a 3-dimensional feature space. [16]

Observe the example in Figure E.1 [16]. The previously mentioned features are represented in that space: `wings`, `sky`, and `engine`. Words are positioned in that space based on how related they are to their characteristics. Thus, *helicopter*, *drone*, and *rocket* are related by `sky` and `engine`, but in a different degree. Analogously to the words in red, *goose*, *eagle* and *bee*. Meanwhile, `jet` can be described with all the features.

Divergence measures between tokens can be computed over embedding spaces, using metrics such as Hamming distance, Manhattan distance, and more recently, cosine similarity.

Due to the hardness of creating a semantic relationship model, pre-trained embedding models are typically used. Two of the most famous and foundational algorithms are *Word2Vec* [41] and *Glove* [47] (published in 2014, when the first s2s models based on RNN appeared). Still, there are other newly models, such as *Mistral* or *MiniLM*.

E.3 Lemmatization

The problem with working with these models (which, in use, act as dictionaries) is that to understand the entire text and its context they require knowing all the words or subwords involved. This means having to use very large distributed representations, which can slow down the text conversion, generation, and processing process. For this reason, a

process known as lemmatization is usually used before vectorization.

Lemmatization is a linguistic process that consists of finding the root or base word form, known as a *lemma*, from different inflected forms (i.e., grammatical variations) that a word can take. This reduces the form of a word with the same meaning that can be presented and allows dictionaries to process each embedding or token faster. See Figure E.2.

In the NLP field, lemmatization process is described as follows: [28]

1. **Tokenization**: division of text into smaller blocks (words or sub-words in this case).
2. **Morphological analysis**: The morphological characteristics of words are identified, such as the root, prefix, suffix, etc.
3. **Assignment of grammatical categories**: The grammatical category (noun, verb, adjective, etc.) of each word in the context is determined.
4. **Lemma identification**: Using linguistic rules and/or dictionaries, the corresponding lemma is searched for each word according to its grammatical category and its inflected form. For example, for the verb "drinking", the lemma would be "drink"; for the noun "dogs", the lemma would be "dog".
5. **Normalization**: All instances of a word are replaced by its corresponding lemma.

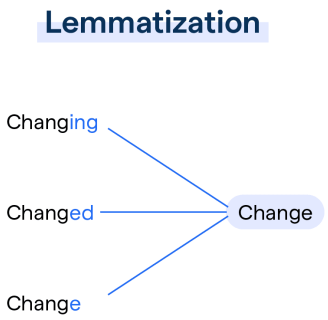


Figure E.2: Some lemmatization examples. [7]

NOTE: stemming can be used alternatively to lemmatization, but it has less accuracy since it can invent words during the tokenization process.

Appendix F

Transformer position encoding

Let t the desired position, in an input sentence, $\vec{p_t} \in \mathbb{R}^d$ its corresponding encoding and $d = d_{model}$ the dimension of the embedding vector space. Then, for each embedding position, the position encoding $g : \mathbb{N} \to \mathbb{R}^d$ is given by the expression:

$$\vec{p_t}^{(i)} = g(t)^{(i)} = \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases} \tag{F.1}$$

In the original paper [61], $\omega_k = \frac{1}{10{,}000^{i/d}}$. Notice that period is given by $T_k = \frac{2\pi}{\omega_k}$.

F.1 Absolute position encoding

Consider the following example: Suppose an embedding model with dimension $d = 6$ and maximum token length $pos_{max} = 10$. Now, as extracted from the expression F.1, the even indices i will be encoded by the sine function; meanwhile, the odd indices by the cosine function. If both indices are stacked based on the position and sorting indices, results from Table F.1 are obtained.

In the case of one-hot encoding, determining the relative position of elements is straightforward by examining the bit patterns. Similarly, with sinusoidal encoding, understanding which position supersedes another becomes clear upon analyzing the decreasing period pattern. See Figure F.1.

F.2 Relative position encoding

In 2017 Google's paper [61], asses the following statement: «We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of

Table F.1: Positional Embeddings. Extracted from own-made Python script.

Position	Index 0	Index 1	Index 2	Index 3	Index 4	Index 5
0	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000
1	0.8415	0.5403	0.0464	0.9989	0.0022	1.0000
2	0.9093	-0.4161	0.0927	0.9957	0.0043	1.0000
3	0.1411	-0.9900	0.1388	0.9903	0.0065	1.0000
4	-0.7568	-0.6536	0.1846	0.9828	0.0086	1.0000
5	-0.9589	0.2837	0.2300	0.9732	0.0108	0.9999
6	-0.2794	0.9602	0.2749	0.9615	0.0129	0.9999
7	0.6570	0.7539	0.3192	0.9477	0.0151	0.9999
8	0.9894	-0.1455	0.3629	0.9318	0.0172	0.9999
9	0.4121	-0.9111	0.4057	0.9140	0.0194	0.9998

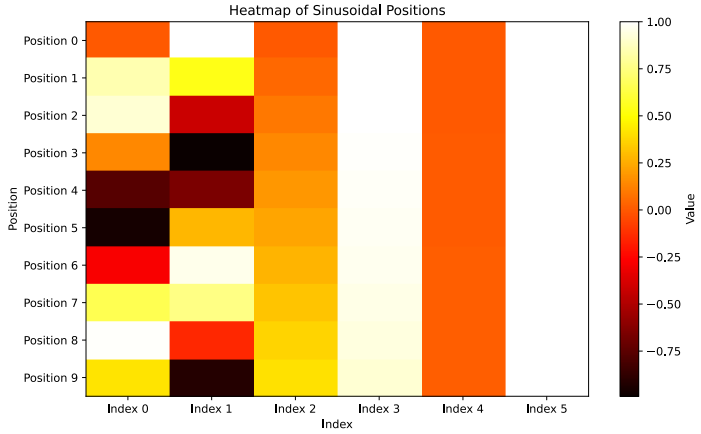


Figure F.1: Positional Embedding representation.

PE_{pos}.»

Indeed, for every sine-cosine pair with frequency ω_k there is a linear transformation $M \in \mathbb{R}^{2\times 2}$ (not dependant of t) such that:

$$M \cdot \begin{bmatrix} \sin(\omega_k \cdot t) \\ \cos(\omega_k \cdot t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k \cdot (t+\phi)) \\ \cos(\omega_k \cdot (t+\phi)) \end{bmatrix} \tag{F.2}$$

Therefore, for some positions, $\vec{p}_t \equiv \vec{p}_{t+1}$. So, effectively, the model can learn to attend by relative positions, because those vectors are well-known in previous or subsequent positions.

Additionally, another interesting property of these functions is since they are bounded and continuous functions, the distance between neighboring time steps is symmetrical and decays nicely with time.

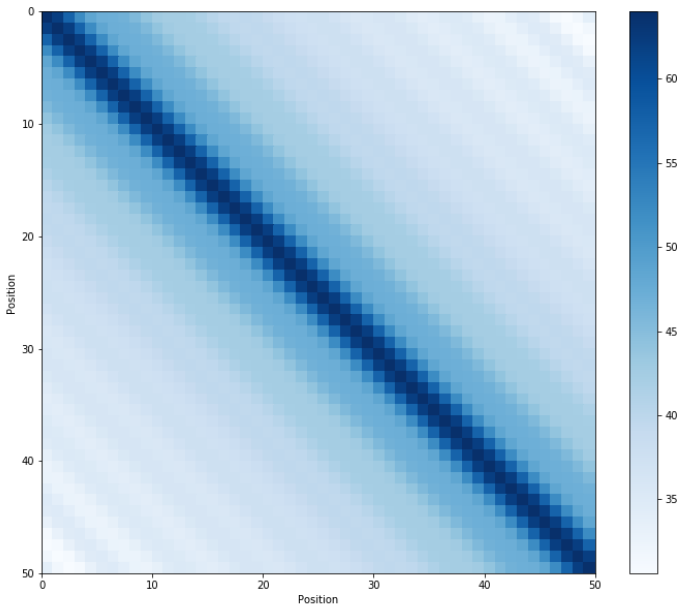


Figure F.2: Dot product of position embeddings for all time-steps. Source: [27]

Appendix G

Responses and Context example

Context	Original Response	Base Model	Finetuned Model
I know that I need to get past my feelings for this person I fell in love with, but it's so difficult to move on because he showed me feelings I've never felt before. I feel like I don't want to be without my genuine love for him, but logically, I know I need to be without him. I can't discuss this with anybody in my life because the conditions surrounding our love are considered "wrong" by a lot of people. How can I get myself to just move on?	Love is a deep and nuanced feeling so start to accept that a simple cutting of emotional connection isn't possible. Honor the love you felt for this person. Acknowledgement of your love feelings will most likely help you find a dignified way to accept them while at the same time slowly developing new ways to live your life in a way which does not include the actual person. Your feelings will always belong to you. The newly discovered sense of how love feels will be with you and positively influence all your relationships. There is a benefit to feeling loved even if it is not forever in this lifetime. Also, your partner opened your feelings, the feelings which showed in fact already were within you. He opened the door and this door belongs to you. I hope you will find ways to appreciate having been loved and to be open that your feelings of being loved are a positive influence in all your relationships.	Firstly, it's important to acknowledge that it's okay to feel the way you do. It's okay to feel conflicted and confused, and it's okay to take time to figure things out. Don't be too hard on yourself, and try not to beat yourself up over your feelings. One thing you could try is to take some time and space for yourself. Sometimes, when we're feeling overwhelmed or stuck, taking a step back and giving ourselves some time and space can help us gain some perspective. Try taking a few days or a week to focus on yourself, your own needs, and your own well-being. Engage in activities that make you happy, and try to do things that bring you joy and fulfillment. Another thing you could try is to talk to someone you trust. This could be a close friend, a family member, or a mental health professional. Talking through your feelings with someone who is non-judgmental and supportive can help you process your emotions and gain some clarity. Just make sure that you choose someone who is someone you feel comfortable talking to, and who will not judge you or try to tell you what to do. It's also important to remember that you don't have to do this alone. Reaching out for support from others can be incredibly helpful, whether it's through talking to a friend or family member, or seeking out professional help. Remember that you don't have to go through this alone, and that there are people and resources available to help you. Finally, try to be kind to yourself. Remember that it's okay to take time to figure things out, and that it's okay to feel confused or uncertain. Be patient with yourself, and try to practice self-compassion. Remember that you are doing the best you can, and that that is enough. In conclusion, moving on from feelings for someone can be incredibly difficult, especially when the conditions surrounding your love are considered "wrong" by a lot of people. However, it's important to remember that you don't have to do this alone, and that there are people and resources available to help you. Take some time and space for yourself, talk to someone you trust, and remember to be kind to yourself. With time and support, you can move on from these feelings and find happiness and fulfillment in other areas of your life.	It's important to acknowledge and validate your feelings. It's okay to feel sad, confused, and lost after a breakup, especially when it's not what you wanted or expected. Allow yourself to feel these emotions and don't try to suppress them. Instead, try to channel them into something productive, like writing, drawing, or exercise. Next, try to focus on the present moment and what you can control. Instead of dwelling on the past or worrying about the future, try to stay grounded in the present. Practice mindfulness techniques, like deep breathing or meditation, to help you stay present and centered. It's also important to take care of yourself during this time. Make sure you're getting enough sleep, eating well, and engaging in activities that bring you joy and fulfillment. Exercise, yoga, or spending time in nature can be great ways to boost your mood and reduce stress. Another helpful strategy is to practice self-compassion. Be kind and understanding towards yourself, and try not to beat yourself up over your feelings. Remember that it's okay to feel hurt and that it's normal to take time to heal. Finally, try to focus on the positive aspects of your life. Think about the things that bring you joy, whether it's spending time with friends and family, pursuing a hobby, or setting new goals for yourself. Focusing on the positive can help shift your perspective and make it easier to move on. Remember, healing takes time, and it's okay to take things one step at a time. Be patient with yourself and allow yourself the space to heal and grow.

Table G.1: Comparison of responses from the source, base, and fine-tuned model given a specific context.

Bibliography

[1] Firas Abuzaid et al. *To Index or Not to Index: Optimizing Exact Maximum Inner Product Search*. 2019. arXiv: 1706.01449 [cs.IR].

[2] Shun-ichi Amari. "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* 5.4-5 (1993), pp. 185–189.

[3] Mohamed Azharudeen. *Rotary Positional Embeddings: A Detailed Look and Comprehensive Understanding*. Medium. Accessed: 13/06/2024. Jan. 2024. URL: https://medium.com/ai-insights-cobet/rotary-positional-embeddings-a-detailed-look-and-comprehensive-understanding-4ff66a874d83.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[5] Satanjeev Banerjee and Alon Lavie. "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments". In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 2005, pp. 65–72.

[6] Nicolas Bertagnolli. *Counsel chat: Bootstrapping high-quality therapy data*. 2020.

[7] *Bot Penguin*. Online; accessed 6-May-2024. URL: https://botpenguin.com/glossary/lemmatization.

[8] Jason Brownlee. *How to Avoid Overfitting in Deep Learning Neural Networks*. https://machinelearningmastery.com/how-to-avoid-overfitting-in-deep-learning-neural-networks/. Accessed: 2024-06-15. Aug. 2019.

[9] Paolo Campolucci et al. "On-line learning algorithms for locally recurrent neural networks". In: *IEEE transactions on neural networks* 10.2 (1999), pp. 253–271.

[10] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[11] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[12] Israel Cohen et al. "Pearson correlation coefficient". In: *Noise reduction in speech processing* (2009), pp. 1–4.

[13] Anirudh Dagar. *Dive Into Deep Learning. 10.1. Long Short-Term Memory (LSTM).* Online; accessed 17-February-2024. 2023. URL: https://d2l.ai/chapter_recurrent-modern/lstm.html.

[14] Deci Research Team. *The Evolution of the Modern Transformer: From 'Attention Is All You Need' to GQA, SwiGLU, and RoPE.* Accessed: 2-June-2024. Feb. 2024. URL: https://deci.ai/blog/evolution-of-modern-transformer-swiglu-rope-gqa-attention-is-all-you-need/.

[15] Dive into Deep Learning. *Gated Recurrent Units (GRUs).* https://d2l.ai/chapter_recurrent-modern/gru.html. Online; accessed: 22-March-2024. 2020.

[16] Guillaume Desagulier. *Word embeddings: the (very) basics.* Online; accessed: 18-February-2024. 2018. URL: https://corpling.hypotheses.org/495.

[17] Tim Dettmers et al. *QLoRA: Efficient Finetuning of Quantized LLMs.* 2023. arXiv: 2305.14314 [cs.LG].

[18] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2019. arXiv: 1810.04805 [cs.CL].

[19] Marcus Frean. "A" thermal" perceptron learning rule". In: *Neural Computation* 4.6 (1992), pp. 946–957.

[20] Bijit Ghosh. *Advanced Techniques for Fine-Tuning LLMs.* Accessed: 2024-06-11. Oct. 2023. URL: https://medium.com/@bijitghosh/advanced-techniques-for-fine-tuning-llms-5min-read.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[22] S Haykin. "Neural Networks and Learning Machines, edit Prentice Hall". In: *New Jersey, USA* (2008).

[23] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models.* 2021. arXiv: 2106.09685 [cs.CL].

[24] Lei Huang et al. *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions.* 2023. arXiv: 2311.05232 [cs.CL].

[25] Laveen N Kanal. "Perceptron". In: *Encyclopedia of Computer Science.* ACM Digital Library, 2003, pp. 1383–1385.

[26] Vladimir Karpukhin et al. *Dense Passage Retrieval for Open-Domain Question Answering.* 2020. arXiv: 2004.04906 [cs.CL].

[27] Amirhossein Kazemnejad. *Transformer Architecture: The Positional Encoding.* https://kazemnejad.com/blog/transformer_architecture_positional_encoding/. Online; accessed 14-April-2024. 2024.

[28] Divya Khyani and Siddhartha B S. "An Interpretation of Lemmatization and Stemming in Natural Language Processing". In: *Shanghai Ligong Daxue Xuebao/Journal of University of Shanghai for Science and Technology* 22 (Jan. 2021), pp. 350–357.

[29] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980.

[30] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[31] Towards Data Science Simeon Kostadinov. *Understanding Encoder-Decoder Sequence to Sequence Model*. https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346. Online; accessed: 28-March-2024. 2019.

[32] Taku Kudo and John Richardson. "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing". In: *arXiv preprint arXiv:1808.06226* (2018).

[33] Hugo Larochelle et al. "An empirical evaluation of deep architectures on problems with many factors of variation". In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 473–480.

[34] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL].

[35] Mu Li et al. "Efficient mini-batch training for stochastic optimization". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 661–670.

[36] Pengfei Liu et al. "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing". In: *ACM Computing Surveys* 55.9 (2023), pp. 1–35.

[37] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. arXiv: 1608.03983.

[38] Anqi Mao, Mehryar Mohri, and Yutao Zhong. *Cross-Entropy Loss Functions: Theoretical Analysis and Applications*. 2023. arXiv: 2304.07288.

[39] Larry R Medsker and LC Jain. "Recurrent neural networks". In: *Design and Applications* 5.64-67 (2001), p. 224.

[40] Meta. *Introducing Meta Llama 3: The most capable openly available LLM to date*. Meta AI. Accessed: 2024-06-17. Apr. 2024. URL: %5Curl%7Bhttps://ai.meta.com/blog/meta-llama-3/%7D.

[41] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[42] *Multi-Head Attention*. Online; accessed 27-April-2024. URL: https://d2l.ai/chapter_attention-mechanisms-and-transformers/multihead-attention.html.

[43] Graham Neubig. "Neural machine translation and sequence-to-sequence models: A tutorial". In: *arXiv preprint arXiv:1703.01619* (2017), pp. 39–42.

[44] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. "A review on the attention mechanism of deep learning". In: *Neurocomputing* 452 (2021), pp. 48–62. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.03.091. URL: https://www.sciencedirect.com/science/article/pii/S092523122100477X.

[45] Franz Josef Och and Hermann Ney. "Statistical machine translation". In: *5th EAMT Workshop: Harvesting Existing Resources*. 2000.

[46] Farty Pants. *What rank (r) and alpha to use in LoRA in LLM fine-tuning?* Accessed: 12-June-2024. Nov. 2023. URL: https://medium.com/@fartypantsham/what-rank-r-and-alpha-to-use-in-lora-in-llm-1b4f025fd133.

[47] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[48] Mary Phuong and Marcus Hutter. "Formal algorithms for transformers". In: *arXiv preprint arXiv:2207.09238* (2022).

[49] Marius-Constantin Popescu et al. "Multilayer perceptron and neural networks". In: *WSEAS Transactions on Circuits and Systems* 8.7 (2009), pp. 579–588.

[50] Wisam Qader, Musa M. Ameen, and Bilal Ahmed. "An Overview of Bag of Words; Importance, Implementation, Applications, and Challenges". In: June 2019, pp. 200–204. DOI: 10.1109/IEC47844.2019.8950616.

[51] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[52] Sebastian Raschka. "Practical Tips for Finetuning LLMs Using LoRA (Low-Rank Adaptation)". In: *Ahead of AI* (Nov. 2023). URL: https://sebastianraschka.substack.com/p/practical-tips-for-finetuning-llms.

[53] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks". In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[54] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. "Activation functions in neural networks". In: *Towards Data Sci* 6.12 (2017), pp. 310–316.

[55] Noam Shazeer. *GLU Variants Improve Transformer*. 2020. arXiv: 2002.05202 [cs.LG].

[56] Amine Tadjer, Aojie Hong, and Reidar Bratvold. "Machine learning based decline curve analysis for short-term oil production forecast". In: *Energy Exploration & Exploitation* 39 (May 2021), p. 014459872110117. DOI: 10.1177/01445987211011784.

[57] Hong Hui Tan and King Hann Lim. "Vanishing gradient mitigation with deep learning neural network optimization". In: *2019 7th international conference on smart computing & communications (ICSCC)*. IEEE. 2019, pp. 1–4.

[58] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[59] UN General Assembly. *Transforming our world: the 2030 Agenda for Sustainable Development*. https://www.refworld.org/legal/resolution/unga/2015/en/111816. Accessed: 2024-06-16. Oct. 2015. URL: https://www.refworld.org/legal/resolution/unga/2015/en/111816.

[60] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[61] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[62] Sun-Chong Wang and Sun-Chong Wang. "Artificial neural network". In: *Interdisciplinary computing in java programming* (2003), pp. 81–100.

[63] Qingsong Wen et al. "Transformers in time series: A survey". In: *arXiv preprint arXiv:2202.07125* (2022).

[64] Thomas Wolf et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. 2020. arXiv: 1910.03771 [cs.CL].

[65] Jia Wu et al. "Hyperparameter optimization for machine learning models based on Bayesian optimization". In: *Journal of Electronic Science and Technology* 17.1 (2019), pp. 26–40.

[66] Lingling Xu et al. *Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment*. 2023. arXiv: 2312.12148 [cs.CL].

[67] Yong Yu et al. "A review of recurrent neural networks: LSTM cells and network architectures". In: *Neural computation* 31.7 (2019), pp. 1235–1270.

[68] Jingzhao Zhang et al. "Why gradient clipping accelerates training: A theoretical justification for adaptivity". In: *arXiv preprint arXiv:1905.11881* (2019).

[69] Tianyi Zhang et al. *BERTScore: Evaluating Text Generation with BERT*. 2020. arXiv: 1904.09675 [cs.CL].

[70] Fuzhen Zhuang et al. "A comprehensive survey on transfer learning". In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.