1 2 9 0

UNIVERSIDADE Ð COIMBRA

Pedro Duarte Santos Henriques

Augmenting Large Language Models with Context Retrieval

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, advised by Prof. Dr. Nuno Lourenço and Dr. Filipe Assunção and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

January 2024

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Pedro Duarte Santos Henriques

Augmenting Large Language Models with Context Retrieval

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, advised by Prof. Dr. Nuno Lourenço and Dr. Filipe Assunção and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

January 2024

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA

FACULDADE DE
CIÊNCIAS E TECNOLOGIA

UNIVERSIDADE Ð
COIMBRA

Pedro Duarte Santos Henriques

Augmenting Large Language Models with Context Retrieval

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Sistemas Inteligentes, orientada pelo Prof. Dr. Nuno Lourenço e pelo Dr. Filipe Assunção e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Janeiro 2024

Abstract

In the recent years, Artificial Intelligence has witnessed significant advancements, particularly in Natural Language Processing. The breakthroughs in Large Language Models (LLMs) have showcased remarkable capabilities in understanding and generating human-like text. However, challenges arise when dealing with large context sizes, both in terms of computational costs and model performance.

In the context of OutSystems, the company is trying to use these models to improve the user experience, by providing automatic code suggestions to developers in the Service Studio platform. With the increasing complexity of applications, models struggle to handle such context sizes, both in terms of computational complexity as in model architectural limitations.

With this in mind, this research aims to address the above issues by investigating the impact of context size on Large Language Models accuracy and performance. The objectives include analysing the gains and drawbacks faced by these models when dealing with large context sizes, and proposing a context retrieval approach to mitigate these effects.

The document structure encompasses a comprehensive background on LLMs and context retrieval techniques, as well as the current work undertaken. The proposed approach outlines steps to address the research questions, and the work plan delineates the tasks for the remaining development period. The document concludes with insights gained from the initial semester and outlines the next steps in this research endeavour.

Keywords

Artificial Intelligence, Large Language Models, Code Generation, Context Size, Context Retrieval

Resumo

Recentemente, a inteligência Artificial assiste a bastantes avanços, principalmente no campo de processamento de linguagem natural. Os avanços em grandes modelos de linguagem têm demonstrado capacidades memoráveis no entendimento e geração de texto. No entanto, desafios emergem quando se tenta lidar com maior quantidade de contexto, seja em termos de custos computacionais, seja em termos de desempenho.

No contexto de Outsystems, a empresa está a tentar utilizar estes modelos para melhorar a experiência de utilizador, ou providenciar sugestões de código para projetos na plataforma Service Studio. Com o aumento de complexidade das aplicações, os modelos têm dificuldades em processar tamanhos de contexto superiores, tanto em termos de complexidade computacional, como em limitações arquiteturais do modelo.

Tendo isto em mente, este trabalho tenta combater estes problemas investigando o impacto de aumentar o contexto, bem como formas de mitigar os efeitos negativos inerentes. Os objetivos do trabalho incluem analisar os ganhos e percas inerentes a esta tarefa, e propor um sistema de capacitação do modelo baseado na adição de um sistema de recolha de contexto.

O presente documento inclui um background teórico em grandes modelos de linguagem e técnicas de recolha de contexto, assim como o trabalho prático já efetuado durante o semestre. A abordagem proposta descreve passos para abordar as questões de investigação e o plano de trabalho delineia as tarefas para o período de desenvolvimento restante. O documento conclui com intuições obtidos no semestre inicial e descreve os próximos passos neste pesquisa.

Palavras-Chave

Inteligência Artificial, Grandes Modelos de Linguagem, Geração de código, Tamanho de Contexto, Recolha de Contexto.

Contents

Acronyms

AI Artificial Intelligence.

ALiBi Attention with Linear Biases.

ANN Artificial Neural Network.

DEI Department of Informatics Engineering.

FCTUC Faculty of Sciences and Technology of the University of Coimbra.

KG Knowledge Graph.

LLM Large Language Model.

LM Language Modeling.

LSTM Long Short-Term Memory.

MSE Mean Squared Error.

NLG Natural Language Generation.

NLM Neural Language Model.

NLP Natural Language Processing.

NLU Natural Language Understanding.

RAG Retrieval-Augmented Generation.

RNN Recurrent Neural Network.

RoPE Rotary Position Embedding.

SKR Self-Knowledge guided Retrieval augmentation.

SLM Statistical Language Model.

SURGE SUbgraph Retrieval-augmented GEneration.

List of Figures

Chapter 1

Introduction

Artificial Intelligence (AI) is a set of technologies that aim to enable computers to perform advanced tasks, while also learning and improving within the process. In one of the first definitions for this concept, [32], Alan Turing asks "Can machines think?" and elaborates a test scenario, where a human interrogator would try to distinguish between a computer and human text response.

From there, various definitions for this concept were proposed, but the constant evolution and changes in computer science make it almost impossible to reach a concrete statement.

Taking the example from Turing's definition, the test is built to archive an AI that can produce cohesive and correct text, but, at the time of the writing, we try to achieve systems that think and act rationally, opening the concept for much more complex tasks.

The rises of researches trying to archive such systems, and searching about ways to give machines the ability to understand and interact using human language, was the origin of a field called Natural Language Processing (NLP).

One of the tasks within NLP is Language Modeling (LM), where the main focus is on creating *"models which can accurately place distributions over sentences, not only encode complexities of language such as grammatical structure, but also distil a fair amount of information about the knowledge that a corpora may contain"* [15].

The release of the transformer architecture, with its capacity to capture long-range dependencies in data as well as parallelization and scalability, has made it possible to train language models with large quantities of data, the Large Language Models (LLMs).

Even though LLMs show major improvements from previous approaches, there still exist some issues. In one hand, the quantity of data that a model can process at a time is limited, or, when it is not, is extremely expensive both to train, and at inference time. On the other hand, their performance depends on the quantity of data that is given to the model as input.

1.1 Objectives and Research Questions

The main objective of this work is to research how to mitigate the issues of LLMs when dealing with large context sizes. To evaluate the proposed mitigation plans, we will be testing in a code assistant problem.

To reach these objectives, some research questions were formulated, in order to further structure the approach to this problem.

- **RQ1 — What are the accuracy gains and performance drawbacks that LLMs face when dealing with large context sizes?** To answer this, an analysis and evaluation of how the context size influences the model performance will be done, in order to identify the risks and benefits of increasing this size.
- **RQ2 — How can a context retriever be built in order to compact the most quantity of information in the least context size?** To answer that, we are going to research and test different approaches to retrieve relevant context and integrate it with the model, as well as researching about what information gives the best performance, while keeping in mind that the context size should be minimized in order to minimize the computational costs.

1.2 Work Plan

This section introduces a high level definition of the work plan in the context of an academic internship at the Department of Informatics Engineering (DEI) from the Faculty of Sciences and Technology of the University of Coimbra (FCTUC) during the academic year of 2023/2024.

The internship is taking place at OutSystems[1], a company founded in 2001 in Lisbon, Portugal, developing a low-code platform which provides tools that can be used by both business and individuals to build applications fast by dragging and dropping visual components into the development environment. With large research funding applied to AI, the company is trying to find ways to adapt and integrate AI models in the platform in order to enhance user experience, with, for example, automatic suggestion of next-best actions in the development of the UI, or logic flows.

In this context, the work plan for the remaining of the internship can be divided into:

- Implement the automatic context retrieval approach
- Analyse the performance of the approach
- Refine the approach

[1]www.outsystems.com

- Write a scientific article with the main results and conclusions
- Write the thesis

1.3 Document Structure

This document describes the work done throughout the first semester, as well as objectives to be achieved by the end of the second semester.

The remainder of this document is organised as follows:

- **Chapter 2** provides an overview of LM concepts, detailing LLM architecture, advantages, and limitations. It also introduces the concept of context retrieval, as well as techniques to mitigate the context length problem.
- **Chapter 3** presents the practical work that have been done throughout this semester.
- **Chapter 4** presents the proposed approach to answer the Research Questions, as well as the work plan for the second semester.
- **Chapter 5** finalizes this document with the main conclusions of the current work.

Chapter 2

State of the Art

This chapter provides an overview of Language Modelling techniques, with special focus on Large Language Models, with detailed overview of its architecture, advantages, and disadvantages. Then it explains and explores state-of-the-art techniques to mitigate the context length limitations of those model, focusing on those related with context retrieval optimization approaches.

2.1 Language Modeling (LM)

In [6] Language Modeling (LM) is defined as *"computational models that have the capability to understand and generate human language"*.

LM main objective is to capture the structure and patterns of a language, enabling the model to generate coherent and contextually relevant text. It functions as a link between two of the most crucial elements of Natural Language Processing (NLP), namely Natural Language Understanding and Natural Language Generation. It accomplishes this by comprehending the input text and generating a meaningful output.

2.1.1 Natural Language Understanding (NLU)

Natural Language Understanding (NLU) is a branch of NLP that focus on capacitating computers with a way to figure out how words connect in a sentence as well as their concrete meaning for that context. Even though for humans this may be a trivial concept, because we do it without even thinking about it, correctly defining processes for letting machines do it can be a difficult task, being reason to a large research focus.

In [19], the authors refer to some of the most common terminologies related to this concept:

- **Phonology:** The study of how the sound can encode the true meaning of a speech.

- **Morphology:** The study of the nature of the words. Considering, for example, the word precaution (pre+caution), it becomes easier to understand that it means a measure taken in advance (pre) to prevent something dangerous (caution).

- **Syntax:** The study of the syntactical dependencies between words.

- **Semantic:** The study of all possible meanings of a sentence by processing its logical structure.

- **Pragmatic:** The study of the knowledge or content that comes from the outside of the document. The real-world knowledge needed to understand what is being talked about in text.

A simple example of how difficult this process can be, is that, in the same sentence, the same word, can have two different meanings, depending on what it is referring to. In the following sentence, the word left has two different meanings inside the same sentence, in the first occurrence, the word left has a similar meaning to leave, whereas in the second, the same word refers to a direction.

I **left** my phone on the **left** side of the room.

NLU can be applied to several use cases, like, build a speech recognition system, or automatically determine the sentiment or emotional tone conveyed in a piece of text.

2.1.2 Natural Language Generation (NLG)

This dimension of NLP aims to find a way for the model to generate coherent text from sets of structured data. To achieve this, the model must be capable of selecting the relevant sections from the input data, as well as structuring and planning the generation correctly.

Similarly to NLU, this task is simple to the human brain, it is capable of generating almost flawless text since an early age, when it isn't even completely developed, but can be a complex challenge for a machine.

The importance of this field is evident when taking the example of chatbots. Many companies use them for initial customer support, leading to immediate, more accessible and efficient customer interactions. But its uses are not limited to this, those systems have a lot of applications like content creation, report generation, or personalized messaging.

In order to understand the concept of LM and what is the influence of Artificial Intelligence (AI) in the referred dimension, we should understand the evolution steps of those models until the current state.

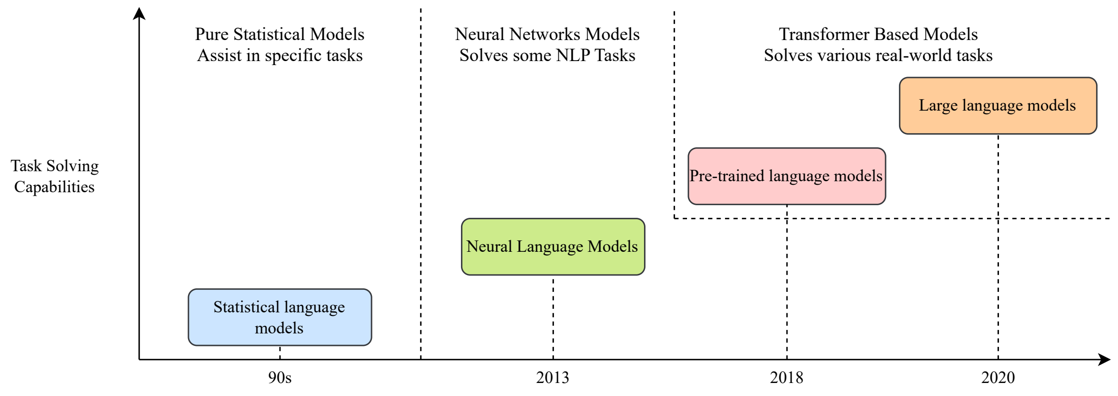

Figure 2.1: Language Modelling evolution in last 30 years (Adapted from [37])

2.1.3 Language Modeling (LM) evolution

In [37] the authors divide the developments of the field into four major development steps (Figure 2.1). In the context of this work, it was decided that it would make more sense to divide the steps into three different categories. Since both Pre-trained language models and Large Language Models are based on the transformer architecture, we decided to merge them in Large Language Models (LLMs), getting the following division:

- Statistical Language Models (SLMs)
- Neural Language Models (NLMs)
- LLMs

It is essential to notice that the borders between each of the development steps aren't strictly defined. In fact, all the presented models are SLM, and LLMs are NLM. That division was applied in the current document, because we understood that it is the best representation of the most relevant improvements in LM context.

As LLMs are strictly related to the thesis context, they will have their own section, in order to deeply explore the concepts related to that topic (Section 2.2).

2.1.4 Statistical Language Models (SLMs)

In [28] a SLM is defined as *simply a probability distribution $P(s)$ over all possible sentences*.

SLM considers *a priori* that a sentence is a formulation of words, so we can build a probabilistic distribution over all possible sentences, s in a language. It also considers a defined set of words, or dictionary, that contains all the possible words. In this context, many SLMs formulate the probability of a sentence $s = W_1.W_2.W_3....W_i$ as:

$$P(s) = \prod_{i=1}^{m} P(w_i | h_{i-1}), \tag{2.1}$$

with $h_i = W_1.W_2.W_3....W_i$ being the history of the studied sentence. The most used SLMs are based on Hidden Markov models and N-Gram Models, so we are going to give a brief introduction of their algorithms.

Hidden Markov models Hidden Markov models were firstly introduced by Leonard E. Baum et al., in a series of statistical papers in the second half of the 60s. [2, 3]

The model assumes that there is an underlying, unobservable (hidden) sequence of states that generates the observed sequence of words. Each word is associated with a particular state, and the transitions between states are governed by probabilities.

Imagining that we want to build a Hidden Markov Model to automatically classify a given set of words lexically, we would need:

- **States (Q):** are the underlying, unobservable entities that the model attempts to model, for example, the word lexical classification (noun, verb, pronoun, . . .)
- **Transition Probabilities (A):** represent the probability of transitioning from state n to state v. It is usually presented in a matrix form, where $A_{n,v}$ can represent, for example, the probability of transition from state Noun to state Verb.
- **Observation Space (O):** are all words that the model can classify. Each word is an observable symbol.
- **Observations (Y):** are the words in the sequence that you want to classify.
- **Emission Probabilities (B):** represents the likelihood of observing a particular symbol (word or observation) given the current hidden state. For example, $b_N(o_t)$ could represent the probability of observing word o_t given that the hidden state is Noun.
- **Initial State Distribution (π):** is an initial probability distribution over states. π_n is the probability that the Markov chain will start in state n. Some states may have $\pi = 0$, meaning that they cannot be initial states.

In order to apply these model and decode the results, the Viterbi Algorithm is applied (Algorithm 1) [17]. It is used to make an inference about the best choice of states so that the joint probability reaches its highest.

Supposing we want to classify the sentence: “I left my phone”. The Hidden Markov model would assign probabilities to different lexical categories for each word, and the Viterbi algorithm would find the most likely sequence of categories for each of these words.

Algorithm 1 Viterbi Algorithm (Adapted from [17])

```
$O \leftarrow Observations\_space$
$Q \leftarrow States$
$\pi \leftarrow Initial\_State\_Distribuition$
$Y \leftarrow Obervations$
$A \leftarrow Transition\_Matrix$
$B \leftarrow Emission\_Probabilities$
for $EachQ : i = 1, 2, 3, ..., K$ do
    $T_1[i, 1] \leftarrow \pi_i \times B_{iy_1}$
    $T_2[i, 1] \leftarrow 0X$
end for
for $EachO : j = 1, 2, 3, ..., T$ do
    for $EachQ : i = 1, 2, 3, ..., K$ do
        $T_1[i, j] \leftarrow max_k(T_1[k, j-1] \times A_{ki} \times B_{iy_j})$
        $T_2[i, j] \leftarrow argmax_k(T_1[k, j-1] \times A_{ki} \times B_{iy_j})$
    end for
end for
$z_t \leftarrow argmax_k(T_1[k, T])$
$x_t \leftarrow s_{z_t}$
for $j = T, T-1, ..., 2$ do
    $z_{j-1} \leftarrow T_2[z_j, j]$
    $x_{j-1} \leftarrow s_{z_{j-1}}$
end for
```

N-Gram Models The n-gram models are based on the assumption that the probability of the next word in a sequence depends only on a fixed size window of previous $n-1$ words. It is a pure-statistical model, used in LM essentially for generation tasks, even though in limited use cases.

In this case, the occurrence probability of a word w_i, in a sentence $s = w_1.w_2.....w_n$, would be given by the equation:

$$P(w_i) = P(w_i|P(w_{i-(n-1)}).P(w_{i-(n-2)})...), \tag{2.2}$$

In conclusion, the selection of the word is based on the combined likelihood that the word will occur together with the previous $n-1$ words.

2.1.5 Neural Language Models (NLMs)

NLM are an implementation of LM with the use of deep learning techniques. Deep Learning is a class of AI algorithms that uses Artificial Neural Networks (ANNs). The most commonly used architectures are Recurrent Neural Network (RNN), and its subtype architecture Long Short-Term Memory (LSTM) .

Artificial Neural Networks (ANNs)

Artificial Neural Networks, is a Machine Learning method inspired by the architecture of the human brain. In the human brain, the cells, called neurons, send information to each other to let humans process data, forming a network.

In [39], the authors state that *compared to a traditional regression approach, the ANN is capable of modelling complex nonlinear relationships. The ANN also has excellent fault tolerance and is fast and highly scalable with parallel processing.*

The ANNs architecture is based on sequentially connected units, also called neurons. These neurons are contained in layers, that are defined by the function they perform in the network (Figure 2.2).

Following an input-to-output sequence, there is:

- **Input Layer:** which represents the inputs or features of the problem.
- **Hidden layer(s):** responsible for learning patterns and representations within the given input data.
- **Output Layer:** produces the final value(s) of the computation that represent the network's prediction or classification for the corresponding input.

In addition to these layers, the connections between neurons play a crucial role in information flow and learning. A neuron's input connection can be the output of a previous layer one, outside data or itself. Each connection has an associated value (weight) to control its influence on the calculations.

Furthermore, each neuron has a parameter to let its value be individually adjusted independently of its inputs (bias). This provides additional flexibility since it allows the neuron to self-adjust its value. Typically, each neuron also has an activation function, to allow non-linearity in the mathematical processing of the data.

We can build a Mathematical representation of each neuron, excluding those on the input layer, by calculating the sum of the weighted inputs, which is the multiplication of each input value (x_i) with its weight (w_i), added to the neuron bias (b) and then apply the activation function (f) to this value.

$$Neuron_{output} = f(\sum_{i}^{n}(x_i * w_i) + b) \tag{2.3}$$

Backpropagation The process of back propagation is used to train ANNs, adjusting the previously referred weights and biases of each neuron. It compares the network's output with the expected result, which can be obtained, for example, from manual calculations or previous examples.

The process can be succinctly explained as follows:

1. **Initialization:** Before the execution, the weights, and bias values of the network are randomly initiated.

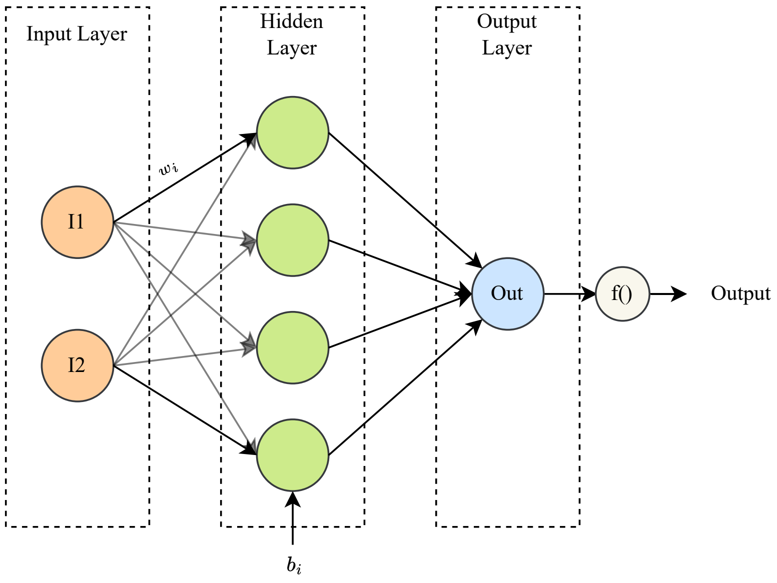

Figure 2.2: Neural Network base architecture

2. **Forward Pass:** The input is fed into the network, and all the computations are made until the output is reached.

3. **Error Calculation:** The comparison of the expected output from the network with the actual output value is called the error, the calculation of this value is done using a loss or cost function, and it is the value that the network should minimize to be trained.

   One example of a loss function is the Mean Squared Error (MSE), so, in a network with n outputs, being Y the expected outputs and $\hat{Y}$ the actual output, the MSE formula is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.4}$$

4. **Backward Pass:** Starting at the output layer, move backwards through the network, calculating the gradient of the loss at each layer and updating respective weights and biases.

   This method requires a learning rate (α) hyperparameter defined in advance, that controls the scale of the updates. If α is too high, the updates will be too big and the model can overshoot the minimum value, failing to convert, not lowering the loss, as expected, but increasing it. On the other hand, if α is too low, the network can require enormous quantities of training iterations, making it computationally expensive.

   The general update rules for a weight(w) and a bias(b) are:

$$Weight : w_{new} = w_{old} - \alpha * \frac{\delta loss}{\delta w_{old}} \tag{2.5}$$

$$Bias : b_{new} = b_{old} - \alpha * \frac{\delta loss}{\delta b_{old}} \tag{2.6}$$

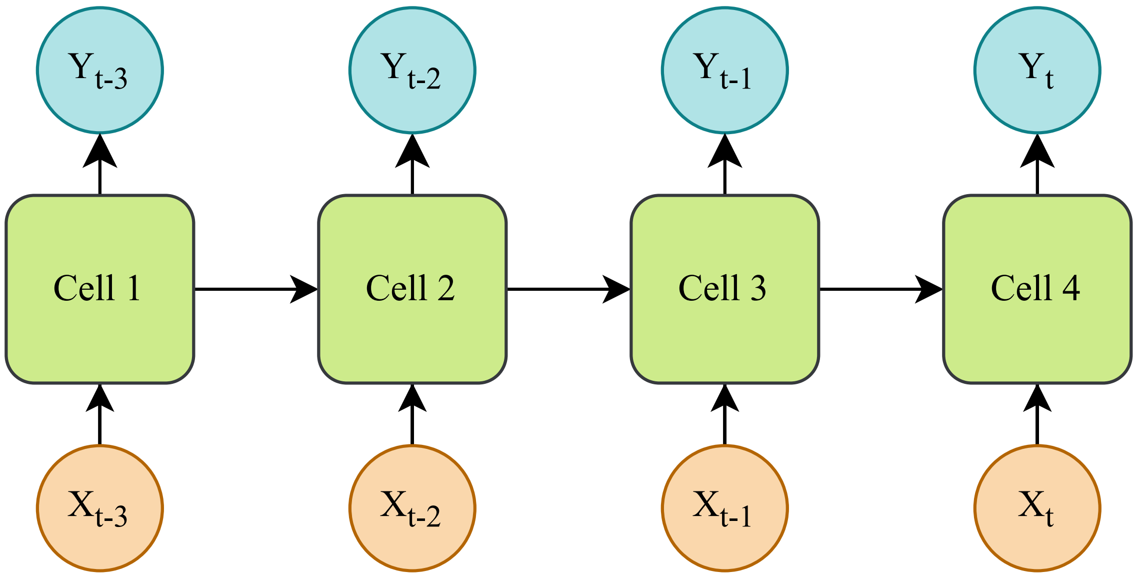

Figure 2.3: Recurrent Neural Network base architecture (Adapted from [1])

5. **Repeat:** The steps 2 to 4 are repeated until a stop criterion is reached. This includes reaching a predefined minimum loss value or completing a predefined number of training iterations.

Recurrent Neural Network (RNN)

RNNs are a type of ANNs which uses sequential or time series data. Being recurrent, means that have a kind of "memory", that keeps track of information from prior inputs, so its outputs at each step also depends on previous inputs.

When the objective is generating or understanding language, this is specially useful, since, in a sentence, let's consider the sentence "I left my phone". All the words have their individual meaning but for them to make sense, it needs to have that specific order.

The RNN structure can be understood as a Neural Network that is replicated at each step of a sequence, called cell. Essentially, the RNN cell is a set of neurons that process input at each time step, as well as information from the previous cell. So each time step will have an input X_t, an *RNNCell* and an output Y_t (Figure 2.3).

To train this type of networks, an extension of Backpropagation, called backpropagation through time, is used. In this algorithm, the loss is calculated to each time step, as well as the gradients, then the weights and biases are updated with the sum of those gradients.

Even though RNNs are a powerful architecture to process sequential data, it has a few challenges associated. One significant problem is the vanishing and exploding gradients.

The vanishing gradient problem occurs when the gradients of the loss function

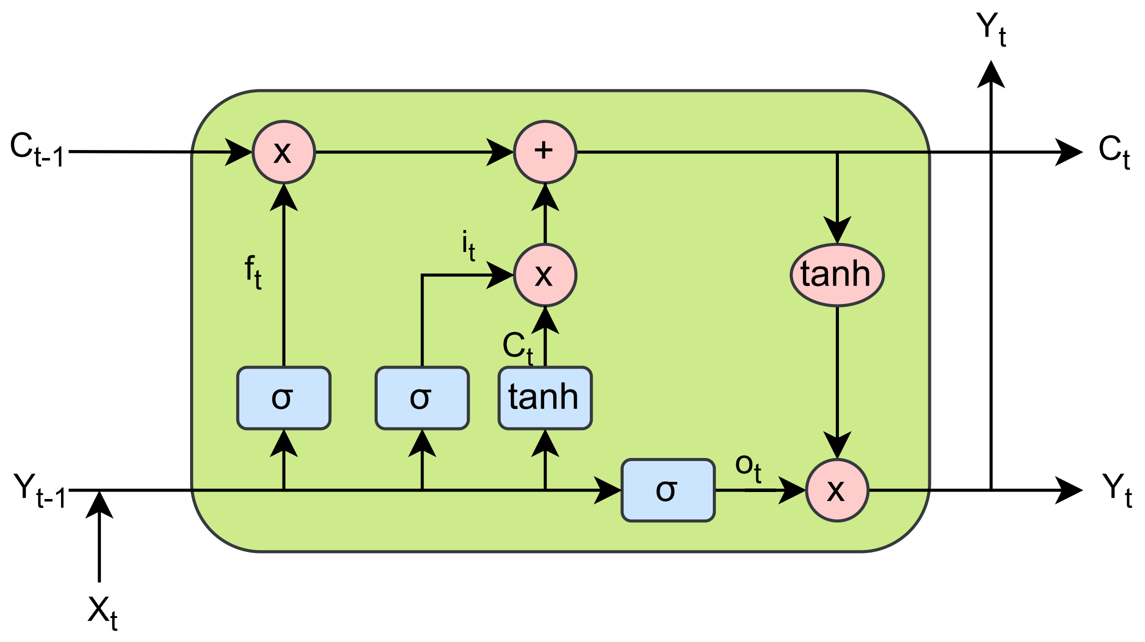

Figure 2.4: LSTM cell architecture (Adapted from [26])

respective to the weights become too small, and, as a result, the weights updates in the training phase will be tiny, reducing the learning capabilities of the model, specially in the long term.

In the order hand, the exploding gradient problem, happens when the gradients are extremely large, leading to convergence issues, as well as unstable training.

Long Short-Term Memorys (LSTMs)

It is a subtype of RNN, developed to decrease the influence of the vanishing gradient problem by modifying the architecture of the RNN cell, adding another value that is added through cells, called the cell state (C_t), and three gates that control the information flow through the network.

1. **Forget Gate (f_t):** Controls the amount of new information to be added to the cell state.

2. **Input Gate (i_t):** Controls the amount of information to be discarded from the cell state.

3. **Output Gate (o_t):** Controls the amount of information to be exposed to the next hidden state.

The sigmoid (σ) activation function is typically used in the gates, to limit their outputs to 0-1 interval. As for activation functions, hyperbolic tangent (*tanh*) is used, to introduce non-linearity and allowing the network to capture more complex relationships in the data (Figure 2.4). «

2.2 Large Language Models (LLMs)

LLMs are a set of LMs based on the transformer architecture [33] and trained on large amounts of data. Some of the most relevant examples include the Open AI GPT (Generative Pre-trained Transformer)[1] family and the Google's BERT (Bidirectional Encoder Representations from Transformers) [8] architecture.

2.2.1 The transformer model

The transformer is an ANN architecture mainly used to do sequence-to-sequence tasks. Introduced in the article "Attention is all you need" [33] surpassed previous NLP methods such as Recurrent Neural Networks in accuracy, training time, and resource utilization, establishing itself as the leading approach in LM. Its architecture is divided into 2 blocks, the encoder, and the decoder (Figure 2.5).

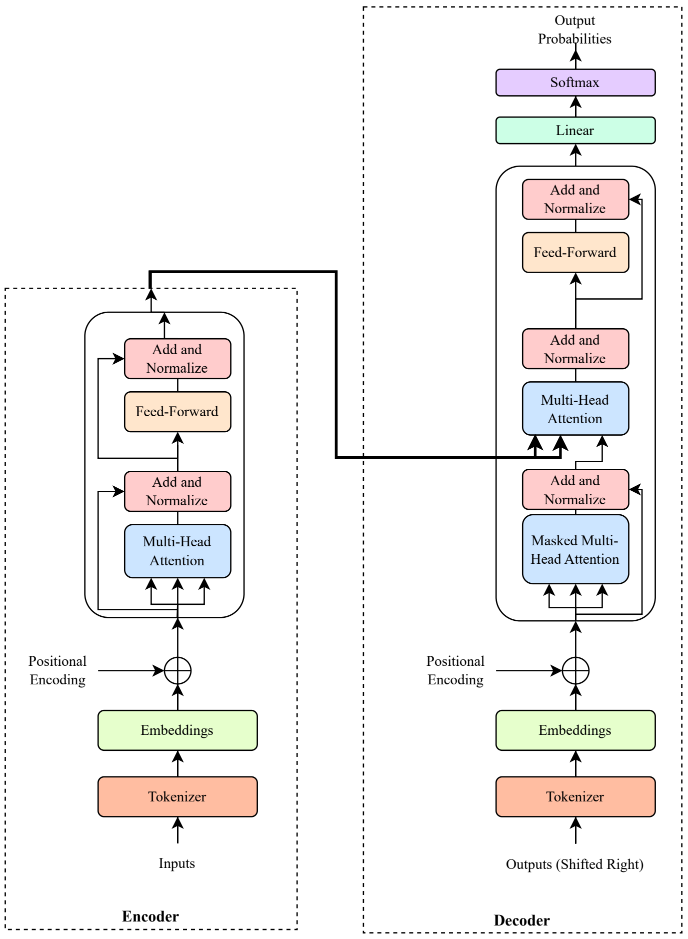

Figure 2.5: The transformer architecture (Adapted from [33])

[1]https://platform.openai.com/docs/models

Encoder Block

Natural Language usually comes in one of two forms, speech or text. In the context of LLM, text is used.

The encoder block is responsible for pre-processing the input text and generating a representation of the meaning and contextual information of the input sequence.

In simple terms, to do this, it uses a process called Tokenization, to split the text into small fractions and then uses a process called embedding to generate a numerical representation of the data. Following this, it adds positional information to this representation and uses a mechanism called attention to find linear word relations, followed by a feed-forward ANN to find non-linear relations in data. All this process is further detailed in the next paragraphs.

Tokenizer The tokenizer is the unit responsible for breaking down text into smaller units used in the model. There are a few Tokenization strategies that can be adopted, depending on the use case [31]. To exemplify this process, the sentence "Tokenizing is good" is going to be used.

- **Character Tokenization** is the simplest form of Tokenization, where each character represents a unique token. So for the example sentence, we will have [T, o, k, e, n, i, z, i, n, g, , i, s, , g, o, o, d].

  In order to process text, this method is not relevant since we cannot find relations between words, only characters, so any text structure is lost.

- **Word Tokenization** is the form of tokenization where each word represents a unique token, then: [Tokenizing, is, my, good]. This method has a clear advantage when compared to the character tokenizer because it maintains the word's characteristics, but it also has disadvantages. As we need to have a unique identifier for each word and the number of different words is huge, it is difficult for the model to keep track of them all. An approach to mitigate this is discarding some less used words, considering them unknown, but then some pieces of information would be missed.

- **Subword Tokenization** is a compromise between the best aspects of character and word tokenization. This means that some more complex words will be divided into prefix and suffix, so the model does not need to have the ability to identify them all. The word "tokenizing", for example, would be divided into "token" and "inizing". This makes it possible for the model to keep the structure of the text while dividing the complex words into simple words.

  In the example, the tokens would be: [Token, ##izing, is, good].

  Those are most commonly the ones used in the Transformer model. For the model to identify the start and end of the sequence, some extra special tokens are usually added. Something like [[start], token, ##izing, is, good, [end]] (Figure 2.7)

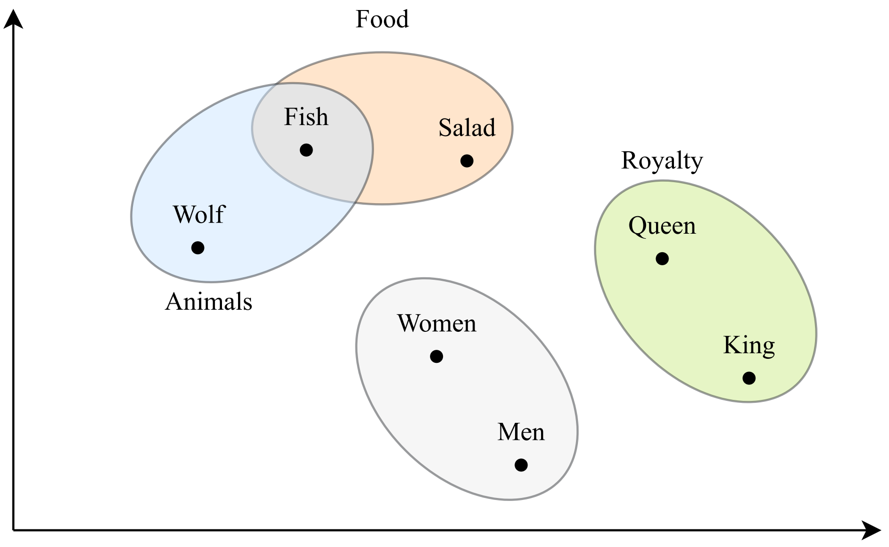

Figure 2.6: Two Dimensional embedding axes with exemplified tokens

Embeddings After doing the tokenization, the model will need a numerical representation of those tokens, this requires the use of embeddings. So, an embedding is a vectorial representation that makes possible the comparison between tokens.

It uses a Dense-Vector representation, this means that the information is compressed into the minimum length possible, and each element of the vector contribute to the overall representation of the object as a whole. The dimension of those vectors varies with the dimensions we need to differentiate an object to another, in the problem context.

Consider the following tokens, in the context of a 2-dimensional embedding representation: King, Queen, Men, Woman, Fish, Salad and Wolf.

In this context, the King is closely related to the Queen, as those concepts are pretty similar. The same happens between the concepts of Men and Women, Fish and Salad, and Fish and Wolf. We can expect, for example, that the distance between Fish and Wolf would not be the same, or in the same direction, as the one from Fish and Salad, even though the token Fish is common in both comparisons, in the first one is related to an animal, whereas in the second is related to food.

Another property of those Embeddings is that they can learn some syntactic and semantic regularities about the tokens, using vector relations. In [24], the authors give the example of the Man-Woman relation being automatically learned by the vectors, as the vectorial calculation of $King - Man + Woman \approx Queen$ (Figure 2.6).

Positional Encoding Positional Encoding is an idea applied in the Transformer model with an intuitive objective, increasing the model's understanding of text flow, by passing the positional information of each token to the model.

- **The sinusoidal approach**, proposed by the authors for the transformer architecture, this approach makes use of sinusoidal functions to create a unique vector for each position. For a given position p and dimension i, the positional encoding will be defined as:

$$PE(p, 2i) = sin(\frac{p}{10000^{\frac{2i}{d}}}) \quad (2.7)$$

$$PE(p, 2i+1) = cos(\frac{p}{10000^{\frac{2i}{d}}}) \quad (2.8)$$

  Here, p represents the position in the sequence, i is the dimension of the positional encoding, and d is the total dimensionality of the embedding. The resulting sinusoidal values are then added to the input embeddings, providing the model with positional information. This is an absolute position approach, meaning that the model will know the position difference from the beginning of the sequence.

- **Rotary Position Embedding (RoPE)** proposed in [30], this is a relative position encoding approach. RoPE makes use of the rotation operation in the Euclidean space and keeps the relative position by rotating the embedding matrix by an angle proportional to its position index.

  The RoPE method can be formulated as follows:

  Let E be the embedding matrix of size $d \times n$, where d is the embedding dimension and n is the number of words in the sentence. Let p be the position index of a word in the sentence, ranging from 0 to $n-1$. The RoPE method rotates the embedding matrix E by an angle θ proportional to the position index p. The rotated embedding matrix is given by $E' = E \times R(\theta)$, where $R(\theta)$ is the rotation matrix in the Euclidean space, in 2d terms, something like:

$$R(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \quad (2.9)$$

  For higher dimensional spaces d, the model can by extrapolated, by dividing the space into $d/2$ subspaces, with the rotation matrix being something like:

$$R(\theta) = \begin{bmatrix} cos(\theta_1) & -sin(\theta_1) & ... & 0 & 0 \\ sin(\theta_1) & cos(\theta_1) & ... & 0 & 0 \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & cos(\theta_{\frac{d}{2}}) & -sin(\theta_{\frac{d}{2}}) \\ 0 & 0 & ... & sin(\theta_{\frac{d}{2}}) & cos(\theta_{\frac{d}{2}}) \end{bmatrix} \quad (2.10)$$

- **Attention with Linear Biases (ALiBi)**, proposed in [27], this is a positional method based on not adding the position encoding vector to the embedding, as the methods seen previously, but instead adds the absolute distance between tokens into the $query * key$ computation at attention phase, as further explained in the document (Section 2.3).

  ALiBi penalises elements further from the current position, but lets the model handle sizes bigger than those who were trained for, whereas the previous methods cannot do it, since are not trained to understand the position embeddings generated for further away positions.

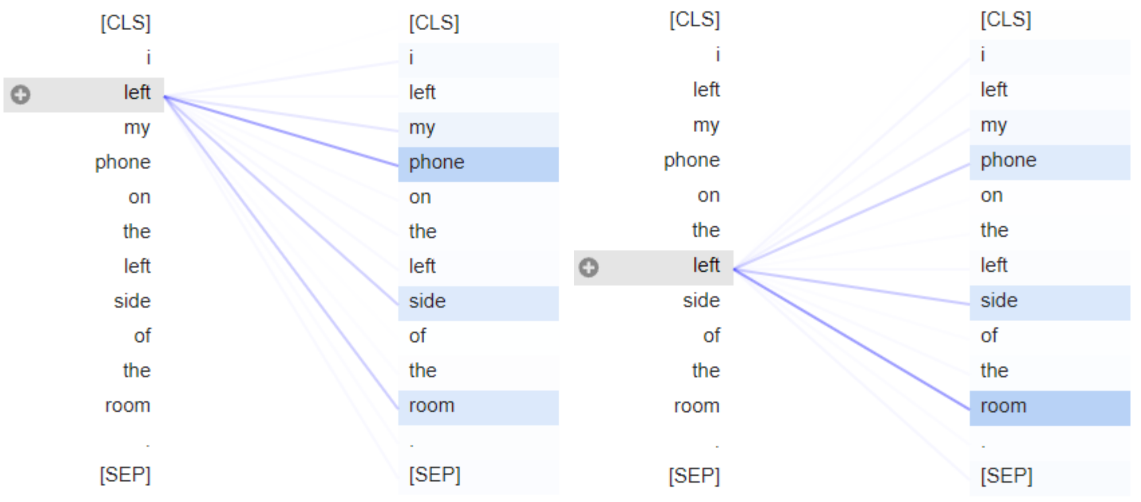

Figure 2.7: Attention applied to "*I **left** my phone on the **left** side of the room.*"

Encoder Layer The Transformer encoder layer is a component of the Transformer model that is responsible for encoding the input sequence into a fixed-size context vector. An encoder block, can have more than one encoder layer. Each encoder layer consists of a multi-head self-attention mechanism and a feed-forward network, which are used to encode the input sequence into a fixed-size context vector.

- **Multi-Head Attention** With the embedding and positional information gathered about the token, the transformer method makes use of the attention mechanism in order to calculate the influence or dependence between values of a sequence.

  The multi-head technique is used, this means that the result of this module should be the concatenation of multiple attention calculations. We can understand this as if the model is calculating the influence or dependence between values in different themes, or categories, so we will have different attention results and then the model will select the relevant information to the context from all those categories.

  Taking the example sentence:

  I **left** my phone on the **left** side of the room.

  Let's analyse the influence of applying attention to the word **left** (Figure 2.7, darker connections mean closer context relation).

  We can see that the first occurrence of the word left is more related to the token of "phone" than to the token of "room" and "side", since it was the phone that was left, with left having a similar meaning to leave, whereas in the second, the left token refers to the left side of the room, and has a stronger connection with those tokens.

  To understand how this works, we need to understand the base Formula used in calculations, so, let's first define the vectors of Query, Key and Value, which are created by multiplying the embedding with model-trained vectors.

- **The Query** is the word/token that we are going to relate with the remaining, in the example, the word **left**.
- **The Key** is every word in the sequence, its features, what it is (a noun, a verb, . . .)
- **The Value** is the semantic information of each word, its meaning.

We condense the queries together in a matrix, to make the calculations faster, so, by multiplying the query by the key, we get the token scores (or Attention weights). In order to make the matrix multiplication possible and speed up the process to a high quantity of batched data, the K vector is transposed, so we get $token\_scores = QK^T$.

As that is a dot product, its values can get arbitrarily large, which can destabilize the training process. To prevent this from happening, the dot product is scaled by $\frac{1}{\sqrt{d_k}}$, where d is the dimensionality of the query and key vectors, so the variance is reduced. Then the values are normalized with a softmax function to ensure all the column values sum to one.

Then we multiply these scores by the values, and we get the attention values, getting the following formula:

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

- **Feed Forward Layer**
  After those attention calculations are complete, values are passed into a fully connected feed-forward layer.

  This feed-forward layer serves several purposes, the main one, is the possibility of understanding non-linear token relations. That is useful and needed because the attention layer only gets linear relations from data, as it only makes linear computations, so a few pieces of information can be lost.

  In some cases, the feed-forward block can also include a dimensionality reduction step prior to the non-linear layer, which helps the model to reduce the number of parameters and complexity while keeping the most important information gathered by the attention layer.

- **Add and Normalization**
  After Attention and Feed Forward, an Add and Normalize block is used. The add represents the addition of the layer inputs to its outputs. The idea here is if the does not make a significant improvement, the information from the input can still flow through the network.

  After that addition, the Normalize part involves layer normalization to the addition referred, to stabilize the training process, so the result at each training step, has a mean of zero and a standard deviation of one.

Decoder Block

The structure of this block closely resembles that of the encoder. Therefore, rather than reiterating the previously discussed context, the differences will be detailed.

The decoder is an autoregressive model built to sequentially generate tokens. It generates tokens by taking as input its own outputs, shifted to the right, one position off from the one the model is currently generating. This ensures that the model is only considering previously generated tokens, and not future ones.

Decoder Layer Similar to the Encoder Block, the decoder block can have several decoder layers, so the model can scale while improving its results. Each decoder layer consists of a masked multi-head self-attention mechanism, a multi-head attention mechanism that attends to the encoder output, and a feed-forward network, as well as adding and normalization steps.

- **Masked Multi-Head Attention**

  As illustrated in Figure 2.5, the decoder layer has a module of Masked Multi-Head Attention. This is the Multi-Head Attention layer, as previously explained, but with a masking mechanism applied.

  The masking mechanism is used to prevent the model from attending to future positions in the input sequence. The mask is initialized with all the values above the negative diagonal set to a large negative number or minus infinite, and then added to the attention scores. This ensures that the model only attends to the positions in the input sequence that have already been processed, and prevents it from attending to future positions in the input sequence.

- **Multi-Head Attention**

  The Multi-Head Attention mechanism is similar to the Multi-Head Self-Attention mechanism used in the Transformer encoder block, but with a different input.

  In the Multi-Head Attention mechanism, the model receives the output of the encoder's representations as input, and then computes the attention scores between the query and key vectors in the input sequence and the encoder output. The attention scores are used to compute the weighted sum of the value vectors, which is used to update the hidden state of the model.

2.2.2 LLM architectures

The transformer architecture is modular, this means that it not strictly mandatory to have an encoder-decoder architecture, like the one presented. In fact, several of LLMs are based on encoder-only or decoder-only architectures.

Encoder-only In the encoder-only configuration, the transformer model is used to encode the input sequence into a fixed-size context vector, which can be used for tasks such as classification, clustering, and search. The encoder-only configuration is particularly useful for tasks that require understanding the meaning of the input sequence, but do not require generating the output sequence.

Decoder-only architecture The decoder-only architecture is used for tasks that require generating the output sequence based on the input sequence, without further context given by the encoder. The decoder-only architecture is particularly useful for tasks such as language modelling, text generation, and machine translation.

Encoder-decoder architecture The encoder-decoder architecture is used for tasks that require generating the output sequence based on the input sequence and the context vector generated by the encoder block. The encoder-decoder architecture is particularly useful for tasks such as machine translation, where the model needs to generate the output sequence based on the input sequence and the context vector generated by the encoder block.

2.2.3 Advantages of LLMs

As mentioned previously, the transformer model has several advantages over previously used techniques.

Parallelization Parallelization, or the ability for the model to process in parallel more data, helps speed up the inference and training times, as the model can be divided into blocks, those blocks can be individually processed on different units. It also allows the model to be trained with more data, resulting in better performance.

Scalability Regarding Scalability, in most of the cases, the performance of the model gets better with the more parameters it has. This requires more training effort as a trade-off for better results, with some models reaching billions of parameters, requiring more computational power both for training, as for inference time.

Transfer Learning Another big advantage of LLMs is the transfer learning capabilities. The transformer model allows for the possibility of fine-tuning an existent model in another dataset, specializing the model into this data without needing to train the model from scratch.

By fine-tuning an existing model on a new dataset, the model can quickly adapt to the new data and improve its performance on the specific task, allowing for

LLMs to be applied in very specific use cases, when are trained in large amounts of general knowledge.

2.2.4 Disadvantages of LLMs

Even though the transformer architecture provided a big step up from previously applied techniques, it also has some problems associated that researchers are trying to mitigate. In this document, the technical disadvantages will be focused, so we will not refer to ethical and environmental known issues, more information about those, can be found on different articles, like [38]

Hallucination Hallucination occurs when the model generates syntactically correct text, that appears to be factually correct, but in fact isn't because the model fabricated this information.

In the survey [13], the authors classify this problem into two different categories:

- **Factuality Hallucination** *emphasizes the discrepancy between generated content and verifiable real-world facts, typically manifesting as factual inconsistency or fabrication.*
- **Faithfulness Hallucination** *refers to the divergence of generated content from user instructions or the context provided by the input, as well as self-consistency within generated content.*

The authors also present different studies to how this is a problem to LLMs, reasoning for this to occur, measuring methods and mitigation techniques.

Limited Transparency The transformer architecture, as ANNs in general, has a major drawback in terms of output transparency, this is, it is hard to define a relation between the model parameters and the final result, making it hard to know where the information was gathered.

2.3 Context Length Limitations

One of the most crucial limitations of LLMs is in context length. The context length defines the memory that the LLM takes into account when making predictions, i.e., the number of tokens considered when doing the attention calculation.

LLMs have context length limitations by two main causes:

1. **Computational Costs** As seen previously, in attention calculation, each of the N tokens of an input sentence is compared to the N other ones, resulting in N^2 comparisons. This means that attention has quadratic complexity, so

having a model with a context length of 512 tokens should be, theoretically, 64 times faster than using a 4096 one when calculating the attention.

Another change is the tokenization and embedding execution time, as the size of the inputs gets larger, it also takes more time to tokenize and embed all the tokens from those inputs.

2. **Positional Encoding** As referred to when presenting the positional encoding techniques, sinusoidal position encoding does not have extrapolation ability, this is, when the model is trained for a defined size, it can't handle a different size at inference time, as it didn't learn how to understand the position encodings of this higher dimensionality.

In [22], the authors also concluded that language model performance degrades significantly when changing the position of relevant information, indicating that models struggle to robustly access and use information in long input contexts, particularly, when the model needs to access information in the middle of the input. This suggests that the solution may not be just increasing the context length barrier, but also find methods to dynamically select relevant information for a given context.

In previous works, various studies have explored strategies for addressing limitations associated with context length. Some of the most significant are presented in the following sections.

2.3.1 Attention with Linear Biases (ALiBi)

As referred to previously, ALiBi, is a position encoding technique that adds the relative position information linearly at inference time.

It does this, by considering only the previous tokens when considering attention, and adding to those a negative bias that gets bigger with the distance for the considered token. This negative bias $m = 2^{-\frac{8}{n}}$, with n being the number of attention heads the model has.

In summary, the token scores with positional information using ALiBi are giving, for a simple four token example by:

$$token\_score = \begin{bmatrix} q_1k_1 & & & \\ q_2k_1 & q_2k_2 & & \\ q_3k_1 & q_3k_2 & q_3k_3 & \\ q_4k_1 & q_4k_2 & q_4k_3 & q_4k_4 \end{bmatrix} + \begin{bmatrix} 0 & & & \\ -1 & 0 & & \\ -2 & -1 & 0 & \\ -3 & -2 & -1 & 0 \end{bmatrix} \times m \tag{2.11}$$

This method allows position encoding for a higher context length, even if the model wasn't trained to do it, with the main drawback being that further away tokens are almost not considered in the result.

2.3.2 Sparse Attention

Sparse Attention is a method of optimizing the attention mechanism to reduce the mathematical complexity of its calculation, by only considering some segments of the inputs, instead of the whole sequence.

Most of the approaches for this problem are static, this means, that the model selects content from the attention matrix with pre-defined patterns.

In [11], a code completion method using static sparse attention is presented, three different kinds of sections are defined:

- **Window Attention** The selection of a few tokens closer to the current token
- **Bridge Attention** Selection of some tokens to keep and have more information about
- **Global Attention** Selection of tokens from the beginning of context, that usually contains global scope information, like definitions, imports, ...

One of the results gathered from this study was that *the sparse models ... have superior performance compared to the non-sparse models ... on both EM and EditSim metrics, and they also maintain a similar inference speed.*

Most of sparse attention implementations, like Big Bird [36] or Longformer Autoregressive [4], follow a similar architecture.

As for dynamic sparse approaches, one of the most relevant is Landmark Attention, further detailed next.

Landmark Attention Introduced in [25], this method aims to optimize the attention mechanism in order to correctly select the relevant tokens in context.

The way it does it, is by add a new kind of token, called landmark, that works as a gate for the attention mechanism to know if it should consider that block or not, dynamically selecting related context.

Let's consider, for example, the tokens [a, b, c, d, e], imagining that we want to calculate the attention for e, and have a block size of 2, where a is related to e, and the remaining tokens are not. With this method, the attention mechanism would check the landmarks, and see that neither c nor d are related to e, so only consider the a and b block in the attention mechanism (Figure 2.8).

Those tokens are added at the end of a pre-defined block size, and the model is trained to give those tokens a representative vector value of the information that those block token contains.

In order to instruct the transformer to utilize landmark tokens, we modify the conventional attention mechanism such that the attention weight for a token depends on the similarity of the query vector with both the key of the token and the

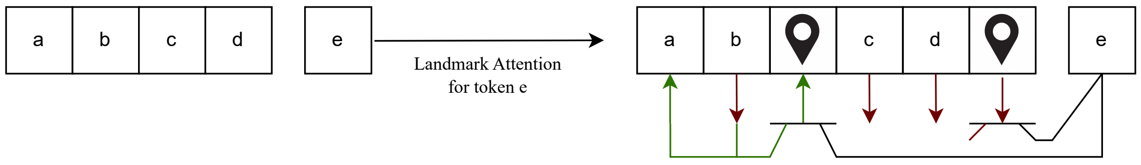

Figure 2.8: Landmark Attention for applied for a group of tokens (Adapted from [25])

key of its block's landmark token, to do this, the Softmax normalization function is replaced by another, named by the authors as *Grouped Softmax*, as formulated:

$$Attention\_Scores(Q,K)_i = GroupedSoftmax(\frac{QK^T}{\sqrt{d_k}}, G_i) \tag{2.12}$$

$$GroupedSoftmax(v,G)_x = \frac{e^{v_x}}{\sum_{y:g_y=g_x} e^{v_y}} \tag{2.13}$$

2.3.3 Context Retrieval

Context Retrieval can help mitigate the context length problem by allowing the language model to focus on specific parts of the context. By retrieving and incorporating relevant information from external sources, the language model can focus on the most important parts of the context, rather than trying to process the entire context at once.

Because of the relevance of this topic in the thesis context, it makes sense for it to have its own dedicated section (Section 2.4).

2.4 Context Retrieval

Context retrieval refers to the concept of retrieving relevant information or concepts based on a specific context or query. In the context of LLMs, a technique called Retrieval-Augmented Generation (RAG) is used. This framework combines retrieval and generation into a single end-to-end model.

RAG was first introduced in [20], used for enhancing the accuracy and reliability of generative AI models with facts fetched from external sources, it relates to the process of accessing and gathering relevant information associated with a particular task or topic. This involves retrieving details from a database, memory, or knowledge base, allowing the system to enhance it capabilities for a given topic.

As seen previously, LLMs have the possibility of being fine-tuned. This essentially is re-training a trained model to fit a specific context. By looking only at this statement, RAG could be considered useless, this is, if fine-tuning can adjust the model to given context, RAG wouldn't be needed. The truth is, a model can

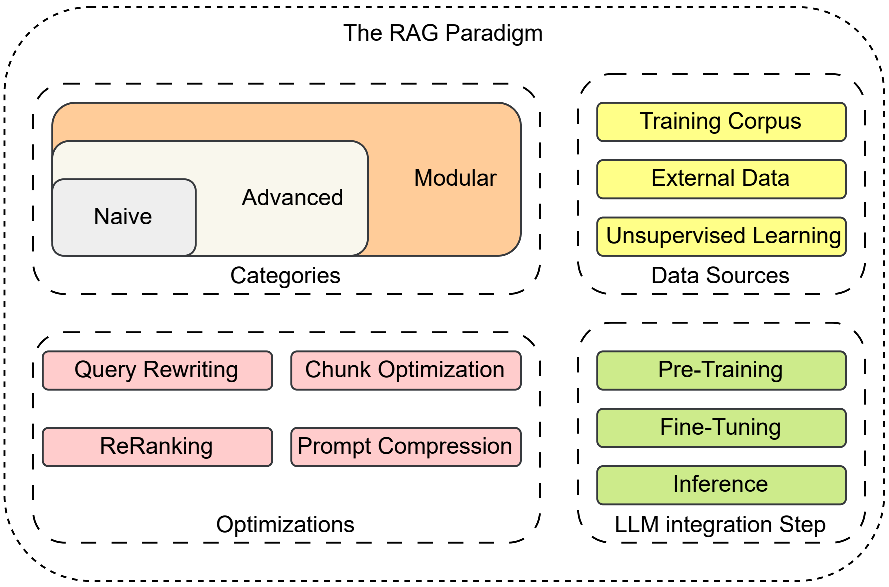

Figure 2.9: RAG current paradigm (Adapted from [9])

be fine-tuned to a given process, and it would possibly have an acceptable performance, but this approach has some drawbacks. The data in which the model is trained is static, that means it requires retraining for knowledge and data updates.

Another draw-back, is that, by only fine-tuning the model, as in inference time it is not clear from where the model got that information, whereas in RAG, we can track it from the information given, providing higher interpretability and traceability.

Instead of the model needing to store all the information in its parameters, some can be saved in this knowledge base or memory and be introduced to the model in runtime. This also makes possible the addition and changes of this knowledge, without having to retrain the model with that new architecture [21].

In [9], the authors provide a deep survey of the current state of RAG, the present RAG paradigm (Figure 2.9), and categorize it into 3 categories, according to their architectures.

- **Naive RAG** classification represents the earliest methodology models in the field of NLP. These models are based on the "read" — "generate" architecture, which involves two main steps: retrieval and generation.

  In the retrieval step, the model reads from a knowledge source, such as a large corpus of text, to gather information related to the provided query. This information is then passed to the LLM, which uses it to generate a response.

  One example of this approach is the REALM model [12], which is capable of augmenting the context given to it by looking at a large corpus, such as Wikipedia. This method allows the model to generate more accurate and relevant responses to queries.

- **Advanced RAG** classification is an extension of the Naive RAG approach, which adds pre-retrieval and post-retrieval steps to further improve performance.

  In the pre-retrieval step, the model aims to optimize the query provided by the user to achieve the best results in the retrieval process. This can involve techniques such as query expansion, where additional terms are added to the query to improve its relevance, or query reformulation, where the query is rewritten to better match the language used in the knowledge source.

  The retrieval step is the same as in the Naive RAG approach, where the model reads from a knowledge source to gather information related to the provided query.

  In the post-retrieval step, the model further refines the retrieved information to improve the quality of the generated response. This can involve techniques such as ReRanking, where the retrieved information is ranked based on its relevance to the query, or fusion, where multiple sources of information are combined to generate a more comprehensive response, as later detailed in this document.

- **Modular RAG** technique presents an alternative to the traditional "read"-"generate" architecture of the Naive RAG approach. Instead of a single, monolithic model, the modular RAG approach consists of multiple, independent modules that can be combined in different ways to achieve the desired functionality.

  A few examples of possible uses are, a search module, where in retrieval steps we can have smaller models to generate searches to also be integrated passed to the LLM, or a memory module, which can store and retrieve information from previous interactions with the user. This can be particularly useful in applications where the user's previous interactions are relevant to the current task, such as in a customer service chatbot.

  One of the key advantages of the modular RAG approach is its flexibility. Because the modules are independent, they can be easily swapped out or modified to suit the specific needs of a particular application.

2.4.1 Data Sources and Representation

The first problem we come across when implementing a RAG framework model is, where to get the data from. In [21], the authors categorize the different sources by three categories:

- **The Training Corpus,** using the training data in inference time in order to gather similar examples in runtime as additional references.

  One way to achieve this is the proposed in [29], a mechanism of Query-Response pairs is used so, at inference time, the responses to similar queries from the one introduced in the model are also provided as model inputs.

- **External Data,** instead of using the training data, a similar architecture from the referred previously can be used, but with external data, this is, data that was not provided to the model to learn.
- **Through Unsupervised Learning**, contrary to the previous methods, does not require a structured architecture. This is based on adding data to the model query, letting the model handle it and figure out what to select from the added context.

Despite those different sources, the data can be further classified by its representation form [9].

- **Unstructured Data**, is the most common form of representation. It can represent a word, tokens, a sentence, a paragraph, or another similar form, that does not follow a defined structure.
- **Structured Data**, as the name suggests, is useful when the data follows a structure. The most common representation is by Knowledge Graphs (KGs), where the data is represented as nodes/entities, and the relationships between different nodes as edges.

2.4.2 Retriever Integration with LLM

Considering those architectures, different approaches of integrating the Retriever and the LLM to build a RAG model can be introduced. One way to relate to them, is by the augmentation stage, this is, the development stage where the retrieval is integrated with the LLM [9]:

Pre-training

One way to integrate the retrieval with the LLM is at pre-training stage. The pre-training stage refers to the initial phase of training a language model on a large dataset to learn general language patterns and structures. This means that the model will trained with more information and, as a consequence, achieve better results with fewer parameters.

An example of this is the RETRO model, based on the GPT-3, that using a BERT Encoder retriever [8], was capable of reaching performances similar from its base model, with 25× fewer parameters [5]. This works, by, splitting the sequence in chunks, whose neighbours, this is, similar information found in the retrieval database, are then encoded and passed in the transformer attention step to the model.

This architecture has, however, some limitations. As the models need to be trained from scratch, this becomes expensive both in training computation costs, as in data.

Fine-Tuning

Another way to integrate the retrieval with the LLM is at fine-tuning stage. As referred to previously, LLMs can be fine-tuned, this essentially is re-train a trained model to fit a specific context.

In most implementations, the Retriever can also be fine-tuned, this means that its outputs can be adapted to match the expected inputs of the LLMs, while those can be tuned to match the specific structure of the problem.

In [16], the authors propose a Knowledge Graph based dialogue system, SUb-graph Retrieval-augmented GEneration (SURGE), built to first retrieve from a KG relevant information.

First, a context-relevant subgraph retriever retrieves the subgraph relevant to the given dialogue history from a knowledge graph.

Specifically, the similarity of the context and triplet embeddings. A triplet embedding is a three-step representation form, in SURGE, should be an entity — relationship — other entity. Then, we encode the retrieved subgraph using the graph encoding.

It encodes these subgraphs though a Graph Neural Network (GNN), as follows:

$$GNN(e_t; G) = UPD(e_t, AGG(e_h | \forall_{e_h} \in N(e_t; G))), \tag{2.14}$$

with $N(e_t; G)$ the neighbour nodes of e_t, AGG a function that aggregates the embeddings of this neighbourhood and UPD a function that updates e_t with the aggregated values.

Finally, contrastive learning is used to enforce the model to generate a knowledge-grounded response with the retrieved subgraph. In contrastive learning, samples are contrasted against each other, and those belonging to the same distribution are pushed towards each other in the embedding space. In contrast, those belonging to different distributions are pulled against each other.

By fine-tuning of both the retriever and generator, we can enhance the generalization capabilities of the models and prevent any potential overfitting that may occur from fine-tuning them separately.

This approach also comes with some drawbacks. Fine-tuning requires large datasets and is computationally expensive, however, this complexity will be potentially smaller when compared to a pre-training stage integration.

Inference Time

The Retriever can also be integrated at inference time. In this case, works almost as a LLM plugin, extending its capabilities without an expensive integration process, because the retriever only extends the context given to the LLM, without it being further adapted.

In [35] this is done through a reward, *with the objective to align the extracted context* *extracted from the previous stage of the generator, ensuring that the text distilled by PRCA* (the developed model) *serves effectively to guide the generator's answering*. It is a process of fine-tuning the retriever to meet the optimal inputs of the LLM.

This approach, beyond being easily integrated and usually archiving better results than using only LLM, without any retriever process, has some drawbacks. With integration levels lower than the previously explained approaches, the retriever can sometimes retrieve useless information, causing a both an increasing of inference computational cost as a decreasing of the LLM accuracy.

2.4.3 Optimization Techniques

In order to improve the results given by RAG, some techniques can be used at different stages to improve retriever performance.

Chunk Optimization

Chunk Optimization is a technique based on dividing the data source documents into smaller chunks, in order to improve the retriever performance.

Most of the time, only a small section of some documents is relevant to be retrieved, so, by dividing those into smaller chunks can make the information selection process more efficient. However, the size of the chunk can deeply affect the performance of the model, so, identifying an optimal size should be a priority, and it depends on several factors, like the context of the application, or the limitations of the embedding model.

Some can be used to optimize this, like sliding windows through the document, or pre-process the document summary or abstract, for example. Those work by optimizing the selection process by carefully choosing information to gather, or selecting the relevance of a document through an overview of it. Another alternative, which is more suitable for multi-hop problems, is graph indexing, which transforms entities and relationships into nodes and connections, improving the relevance of selected documents.

Query Rewriting

Query Rewriting is a technique based on rewriting the user queries, *à priori* of the retrieval process, in order to improve the retrieval process, by providing a more concrete query to search in the retrieval data.

In [23], the authors present a query rewriting approach involving instructing a LLM to modify queries using in-context learning. The training process includes fine-tuning a T5 model as the rewriter to achieve the desired modifications, in order to improve the general context of the query.

ReRank

As referred to previously in this section, ReRanking is a technique based on re-organizing the retrieved documents to prioritize the most relevant ones at the top, and limiting the number of retrieved documents. As limiting the quantity of retrieved documents. This technique aims to reduce the effects of increased context length of LLMs when used with retrievers, while also keeping the most quantity of information possible.

In [18], ColBERT, an adaptation of a Bert Encoder Ranker is presented. The BERT Encoder works as a ranking mechanism by comparing the contents of the document with the contents of the query, where those with closer context should obtain a higher ranking. The main difference of ColBERT with a base BERT is Late Interaction (Figure 2.10).

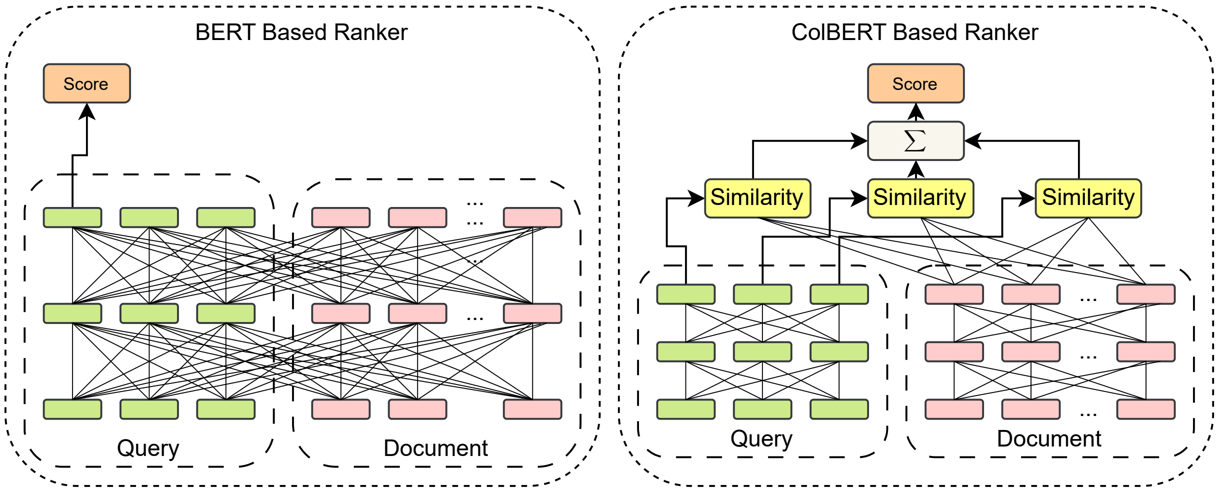

Figure 2.10: Bert vs ColBERT Rankers (Adapted from [18])

Late Interaction is the process of delaying the interaction between queries and documents until the late stages of the model. In simple form, instead of having the BERT encoding every conjunction of document token — query token at the same time, it encodes the query and the document individually, and then it uses similarity metrics to find the most related documents.

Prompt Compression

Prompt Compression is a technique that aims to compress the information introduced to the LLM, with the aim of reducing the number of inputs tokens while keeping the most quantity of information possible.

In [14], the authors divide the prompt compression methods into three different categories:

- Token pruning and/or token merging approaches
- Soft prompt tuning methods

- Information-entropy-based methods

and present LongLLMLingua.

LongLLMLingua presents a prompt compression technique included in the last of those and *designed to enhance LLM's perception of key information (relevant to the question) in the prompt, so that the ... challenge of inferior performance in long context scenarios could be addressed*.

It works by adding an additional LLM, with a relatively low number of parameters (the ones tested in the document have around 7 billion), in order to select and re-order, based on information entropy theory, the most relevant tokens to pass to main LLM. This makes possible a reduction of context length needed to provide the same information, reducing both computational complexity and redundancy.

PoWER-BERT [10] is a technique based on Token pruning, based on exploiting a new type of redundancy within the BERT model pertaining to the word-vectors. Essentially, it uses attention computation, and for each word calculates is significance within the query, this is, the sum of the attention values through the complete query. Then it removes those with the lowest scores, reducing the number of tokens to be processed by the model.

AutoCompressors [7] is a Soft prompt tuning method. Soft prompt tuning is a technique used to fine tune LLMs without changing the existing parameters. It is used in this context to add summarization capabilities to the LLM. Essentially, this approach processes large context by sequentially generate a summarization of small contexts, that is past to the next section.

Self-Knowledge guided Retrieval augmentation (SKR)

Another technique used to improve the performance of RAG models, specifically, used to reduce its computational costs, is Smart Retrieving. Smart Retrieving is a technique based on training the LLM to understand when its self-knowledge is, or it's not enough to answer a specific query.

In [34], the authors propose an approach to implement this by training the LLM and comparing its responses based on self-knowledge with the retrieval augmented responses (Figure 2.11). This makes it possible for the model to really understand what information it has, and what information it needs to retrieve. At inference time, it compares the given query with its training queries, and requires augmentation for those that have a close context with the ones it required at training time.

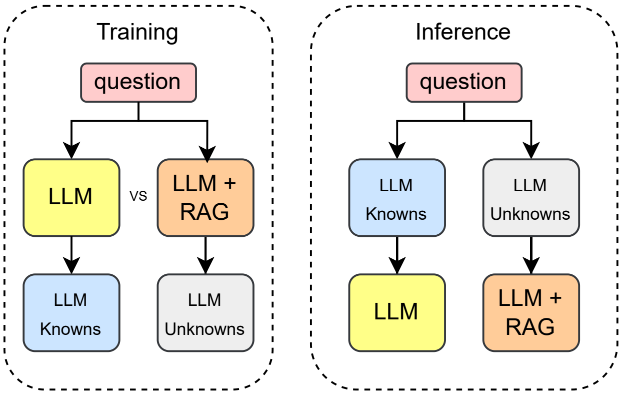

Figure 2.11: SKR architecture (Adapted from [34])

2.5 Summary

This chapter has provided an overview of Language Modeling (LM) techniques, with a focus on Large Language Models (LLMs) and their context length limitations. The chapter has also introduced the concept of Retrieval-Augmented Generation (RAG), which can be used to overcome these limitations by incorporating external information into the language model.

At the beginning, it was discussed the different types of language models, including statistical models, Recurrent Neural Network (RNN), and transformers.

The limitations of LLMs were also discussed; in particular, those related to context length, which can limit their ability to understand and generate long and complex texts.

Some mitigation techniques were introduced, for example, the concept of Retrieval-Augmented Generation as a solution to this problem. RAG can help mitigate the context length problem by allowing the language model to focus on specific parts of the context. By retrieving and incorporating relevant information from external sources, the language model can focus on the most important parts of the context, rather than trying to process the entire context at once.

Chapter 3

Current Work

This chapter presents the work developed during the first semester. The first step of the work, was to understand the LM field, and understand the LLM base architecture. After acquiring enough concepts related to this architecture, an in-depth review of the existing literature on LM, focusing on context length handling techniques, was done.

Moving forward, we began researching about Retrieval Systems and how they are currently being used to augment LLM capabilities, focusing on approaches optimized to reduce/handle context length.

Simultaneously, in order to deeply understand the context of the problem, a understanding of the OutSystems platform and the problem dataset was conducted.

3.1 OutSystems Service Studio

The OutSystems Service Studio is a tool that can be used by both businesses and people to quickly create apps by adding visual components to the development environment. The development environment is then divided into four sections (Figure 3.1):

- **Processes section:** comprises all the elements that allow business processes to be integrated into the application.
- **Interface section:** comprises all UI development of the application.
- **Logic section:** comprises all the logic flows need to the current flow of the application
- **Data section:** comprises the data structure of the application, that is, the entities defined in the application

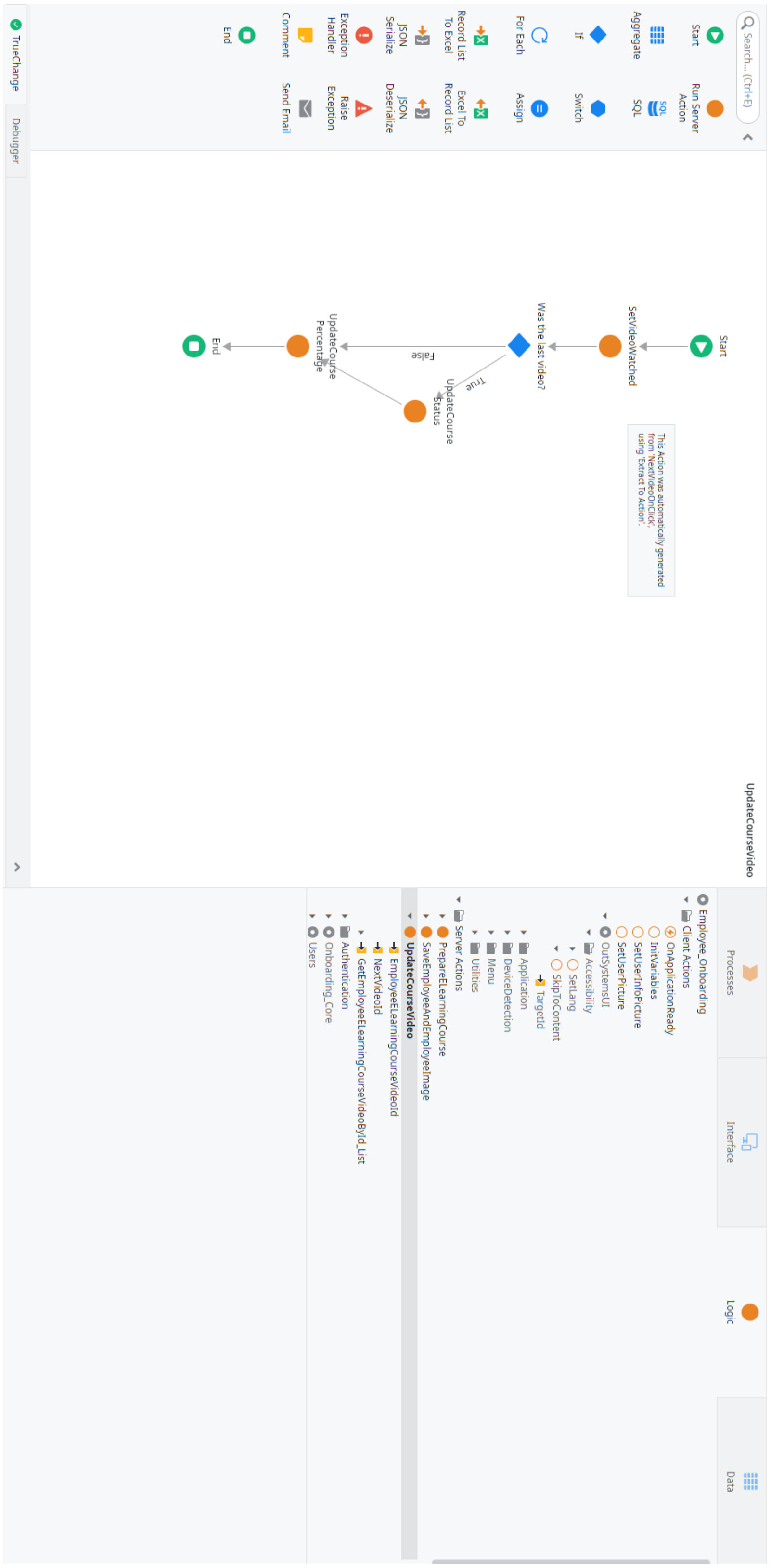

Figure 3.1: Development environment of the OutSystems Platform, showing a previously implemented logic flow

3.2 Dataset

The dataset is a set of generated code representations of the elements in the logic section, previously developed by the OutSystems AI Team. Each element of this section is a flow. A flow is a connected sequence of individual instructions (Figure 3.1). The code representation aims to recreate a Python[1] like code structure.

Each element of the dataset, or each flow, has a field *"text"* with its code representation, that has one or more instructions replaced with a identifier for the model to know where to generate content, that differs with the model used. In the example version, it is the mask token for CodeT5 [2] (*"<extra_id_i>"*, with *i* being the number of the instruction) (Listing 3.1).

It also has the expected value for those instructions in another field, *"label"* (Listing 3.1). Also has some other fields, used for internal identification of the flow in the OutSystems context.

```
### CODE REPRESENTATION ###
from ORD_G_PROCESS_CS.ServerActions import ProcessControl_GetBySwitchNumber, ProcessControlHistory_UpdateProcessHistoryStatus,
        ProcessControl_UpdateProcessStatus, ProcessControlHistory_GetByMessageQueueId
from MARTE_COMMON_BL.ServerActions import ExceptionHandler, Generate_ComputeHash
from (System).ServerActions import CommitTransaction
from ORD_G_PROCESS_BL.ServerActions import ReceiveH1, ProcessControl_SetSwitchDate, Message_GetReferenceInfoQ00,
        ProcessControl_History_SetStatus
from REN_QUEUE_CS.ServerActions import MessageQueuePayload_GetXML, MessageQueue_UpdateStatus, MessageQueue_Update,
        MessageQueue_UpdateComputeHash
from REN_QUEUE_DISPATCHER_BL.ServerActions import MessageOutboundRouting, DispatchOutboundQueueEntries, MessageHandler,
        DispatchInboundQueueEntries, MessageQueue_SetHash, MessageQueue_SetDataQ00, MessageInboundRouting, Validate_MessageQueue,
        Reprocessing_MessageQueue, Reprocessing_SetStatus
from SiteProperties  import TenantId, TenantName, Timeout_REN_Queue_Inbound, Timeout_REN_Queue_Outbound
var1 = IServerAction(Name=CheckDuplicatedPayload, InputParameters={"ComputeHash": str}, OutputParameters={"Valid": bool})
var2 = IStartNode()
var1.add_subflow(var2)
var3 = <extra_id_0>
var2.sequence = var3
var4 = IEndNode()
var3.undefined = var4

### LABEL ###
IAggregateNode(Sources=[MessageQueue], Filters=[(MessageQueue.ComputeHash = ComputeHash)])
```

Listing 3.1: Example of dataset element code representation and respective completion code

3.3 Analysis of Context Length

In order to further understand the context length problem within the dataset, we gathered some statistics about the distribution of the number of tokens in the dataset. Due to the large dataset size, we randomly selected 10% of the dataset instances, to be representative of the whole dataset.

To do the tokenization, and since the dataset is adapted to be used by CodeT5, the tokenizer of this model was used in the current analysis. In this context, we computed the distribution of tokens, for different percentiles. This not only enables gathering information about the average case (50th percentile), but also of the extremes. The distribution is displaying in (Figure 3.2), and enables taking the following insights:

[1]https://www.python.org/
[2]https://blog.salesforceairesearch.com/codet5/

- **25%** of the elements have more than 1500 tokens.
- **50%** of the elements have more than 1730 tokens.
- **75%** of the elements have more than 2110 tokens.
- Considering the limits of CodeT5 model (2048 tokens), we cannot process $\approx$ **26%** of the dataset instances

The analysis of context length distribution is a fundamental step to further develop and understanding of model adaptation and evaluation.

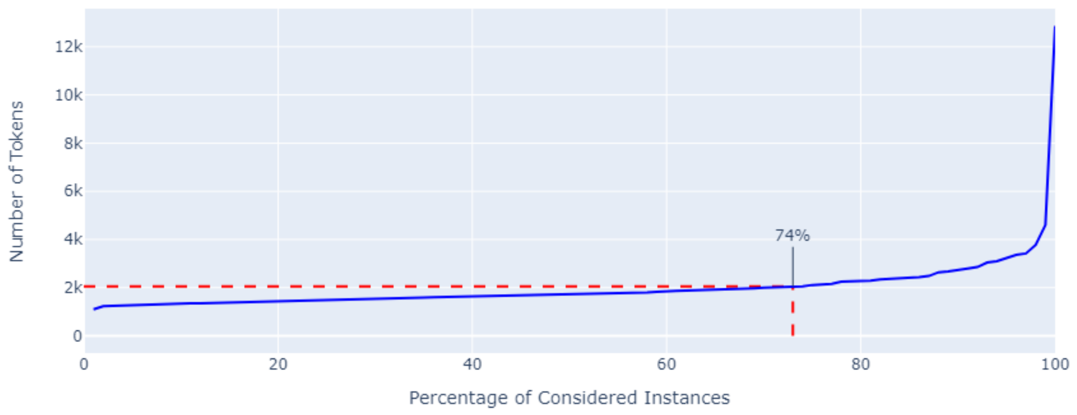

Figure 3.2: Percentile plot for sampled section of the dataset.

Chapter 4

Proposed Approach

This chapter presents the proposed approach for development in the context of the internship. It further details the Objectives and Research Questions (Section 1.1) presented in Chapter 1. As stated there, the aim of the work is to research forms of balancing the computational costs with the LLM performance using a retriever, answering the Research Questions formulated:

- **RQ1 — What are the accuracy gains and performance drawbacks that LLMs face when dealing with large context sizes?**
- **RQ2 — How can a context retriever be built in order to compact the most quantity of information in the least context size?**

A cross-cutting concern of this work, is to evaluate the trade-off between increasing the context given to the LLM system using retrieval augmentation and the performance and increased computational costs associated with large context sizes. In [9], the authors state that *balancing the trade-off between a window that is too short, risking insufficient information, and one that is too long, risking information dilution, is crucial.*

To accomplish that, we decided to formulate some intermediate tasks:

The first task is going to be analysing and choosing computational cost and performance metrics, that will correctly evaluate the model adaptations.

Then, the goal is to fine-tune and evaluate a LLM, without an associated retrieval system, gathering results to be used as a baseline.

Subsequently, we are going to develop a RAG, following a naive approach (Section 2.4), observing the performance changes that comes from the addition of a retrieval to the LLM.

Later in the development, we propose to build a modular RAG implementation, as well as changes in the LLM to handle the increased context. Building and further improving this model is estimated to be the task which will require the most effort to be completed, as it includes studying the feasibility of the proposed optimization techniques, both in terms of RAG, namely Chunk optimization, query rewriting, ReRanking, Prompt compression and SKR (Section 2.4.3),as direct changes to the transformer architecture, namely ALiBi and sparse attention (Section 2.3), followed by their specification and implementation in this context.

Furthermore, a comparison between the referenced approaches should be done, in order to take conclusions about the effects of this work on the computational costs and performance trade-off in the context of LLMs.

The writing of a scientific publication and the final dissertation will take place in parallel with the previous tasks.

Chapter 5

Conclusion

With the increase in uses of LLMs, challenges arise when dealing with large context sizes, both in terms of computational costs and model performance. One of the solutions for this problem is by using a retrieval system to augment the capabilities of this models.

In the augmentation of a LLM using a retrieval system, handling correctly the context used, can be a complex task. The work carried out through the course of this internship will contribute to a deeper understanding of the interplay between retrieval augmentation and context length efficiency.

To accomplish this, we started by researching about LM, focusing deeply on LLM and its limitations in terms of context sizes, as well as some mitigation techniques. Furthermore, we also introduced the concept of Retrieval-Augmented Generation (RAG), presenting and analysing techniques used in previous works to handle large context sizes.

Following the referred research, we analysed some insights about the testing dataset, and proposed an approach to build the retrieval system, detailing the work that need to be done in the second semester.

In conclusion, this document was a result of the work from the current semester, providing background knowledge in terms of LM, LLMs and context retrieval systems, and detailing state-of-the-art approaches to the context length handling problem. A presentation and analysis of the testing dataset was also conducted. Finally, an approach detailing the work plan to the second semester was also developed and presented.

References

[1] Afshine Amidi and Shervine Amidi. Recurrent neural networks cheatsheet star, 2019. URL `https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks`. Last accessed 10 December 2023.

[2] Leonard E. Baum and John A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73:360–363, 1967. URL `https://api.semanticscholar.org/CorpusID:14153120`.

[3] Leonard E. Baum and Ted Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554 – 1563, 1966. doi: 10.1214/aoms/1177699147. URL `https://doi.org/10.1214/aoms/1177699147`.

[4] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

[5] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens, 2022.

[6] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Kaijie Zhu, Hao Chen, Linyi Yang, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *arXiv preprint arXiv:2307.03109*, 2023.

[7] Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts, 2023.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[9] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024.

[10] Saurabh Goyal, Anamitra R. Choudhury, Saurabh M. Raje, Venkatesan T. Chakaravarthy, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating bert inference via progressive word-vector elimination, 2020.

[11] Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. Longcoder: A long-range pre-trained language model for code completion, 2023.

[12] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: retrieval-augmented language model pre-training. *CoRR*, abs/2002.08909, 2020. URL `https://arxiv.org/abs/2002.08909`.

[13] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, 2023.

[14] Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression, 2023.

[15] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[16] Minki Kang, Jin Myung Kwak, Jinheon Baek, and Sung Ju Hwang. Knowledge graph-augmented language models for knowledge-grounded dialogue generation, 2023.

[17] Spyridon Kardakis. *Machine Learning Techniques for Sentiment Analysis and Emotion Recognition in Natural Language*. PhD thesis, School of Engineering Computer Engineering and Informatics Department - University of Patras, 10 2019.

[18] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert, 2020.

[19] Diksha Khurana, Aditya Koli, Kiran Khatter, and Sukhdev Singh. Natural language processing: State of the art, current trends and challenges. *Multimedia Tools and Applications*, 82, 07 2022. doi: 10.1007/s11042-022-13428-4.

[20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[21] Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. A survey on retrieval-augmented text generation. *CoRR*, abs/2202.01110, 2022. URL `https://arxiv.org/abs/2202.01110`.

[22] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

[23] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models, 2023.

[24] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In Lucy Vanderwende, Hal Daumé III, and Katrin Kirchhoff, editors, *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL `https://aclanthology.org/N13-1090`.

[25] Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers, 2023.

[26] Christopher Olah. Lstm networks – colah's blog, 2015. URL `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Last accessed 12 December 2023.

[27] Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *CoRR*, abs/2108.12409, 2021. URL `https://arxiv.org/abs/2108.12409`.

[28] R. Rosenfeld. Two decades of statistical language modeling: where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278, 2000. doi: 10.1109/5.880083.

[29] Yiping Song, Rui Yan, Xiang Li, Dongyan Zhao, and Ming Zhang. Two are better than one: An ensemble of retrieval- and generation-based dialog systems, 2016.

[30] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021. URL `https://arxiv.org/abs/2104.09864`.

[31] L. Tunstall, L. von Werra, and T. Wolf. *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. O'Reilly Media, 2022. ISBN 9781098103248. URL `https://books.google.pt/books?id=pNBpzwEACAAJ`.

[32] A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, 10 1950. ISSN 0026-4423. doi: 10.1093/mind/LIX.236.433. URL `https://doi.org/10.1093/mind/LIX.236.433`.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[34] Yile Wang, Peng Li, Maosong Sun, and Yang Liu. Self-knowledge guided retrieval augmentation for large language models, 2023.

[35] Haoyan Yang, Zhitao Li, Yong Zhang, Jianzong Wang, Ning Cheng, Ming Li, and Jing Xiao. PRCA: Fitting black-box large language models for retrieval

question answering via pluggable reward-driven contextual adapter. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5364–5375, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.326. URL `https://aclanthology.org/2023.emnlp-main.326`.

[36] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences, 2021.

[37] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[38] Terry Yue Zhuo, Yujin Huang, Chunyang Chen, and Zhenchang Xing. Red teaming chatgpt via jailbreaking: Bias, robustness, reliability and toxicity, 2023.

[39] Jinming Zou, Yi Han, and Sung-Sau So. Overview of artificial neural networks. *Artificial neural networks: methods and applications*, pages 14–22, 2009.