ALGERIAN DEMOCRATIC AND POPULAR REPUBLIC
Ministry of Higher Education and Scientific Research
Mohamed Khider University – BISKRA
Faculty of Exact Sciences, Natural and Life Sciences
Computer Science Department

Order N°: SIOD14/M2/2021

Dissertation

Presented to obtain the academic master's degree in

Computer Science

Option: Information Systems, Optimization and Decision (SIOD)

Arabic Text Summarization

By:

RECHID KENZA

Defended on 06/26/2022 before the jury composed of:

Sahli Sihem	MAA	President
Meadi Mohamed Nadjib	MCB	Advisor
Ben Dahmene Asma	MAA	Examiner

Academic year 2021-2022

Acknowledgements

Praise and thanks be to Allah, a great thank for his blessings, grace, giving me good health and the determination to accomplish this work, and he gave me the strength and patience to endure all difficulties.

I sincerely thank my professor and supervisor, ***Meadi Mohamed Nadjib****, who has not limited his valuable advice, useful contributions and all the features that have made a clear impression on the pages of this work.*

All thanks and gratitude to my father, who believed in me and continued to support me from the first day until this moment.

My soul twin Ferial, I wouldn't have gotten that far without you.

My heartfelt thanks to my stepmother H.N. who taught me that studying is a sacred thing and my aunt D.N. who supported me a lot in this academic journey.

My brothers and sisters Aymen, Amdjed, Ridha, Amina and Rehem, I extend my special thanks to you for encouraging me to continue moving forward.

Mom! I have reached my graduation day thanks to all people I have mentioned. May Allah have mercy on you and make your status heaven.

English teacher A.Asma, it is thanks to you, that I wrote the words of this dissertation.

I also do not forget to express my gratitude to those who helped me accomplish this work, especially Khadidja, Samira, Rania, Raid, Youcef and Mohamed.

RECHID KENZA

ABSTRACT

Summarizing the text, which is usually extractive or abstractive, has become necessary due to the exponential amount of textual content and various archives of news, scientific papers, legal documents, etc.

This theme proposes an abstractive summarization of the Arabic text using the XL-Sum dataset, by applying deep learning methods, and using pre-trained models. We offer two systems for summarization. The first of which is based on deep learning approaches. We use a pre-trained model AraBART of the Arabic language and fine-tune it for the task of summarization. The second is the application of the PEGASUS, the English summarizer, to the Arabic text. In addition, we evaluate their effectiveness.

Key words: Automatic text summarization, Extractive, Abstractive, Arabic language, NLP, Deep learning, Word embedding, BERT, BART, PEGASUS.

Résumé

Le résumé du texte, qui est généralement extractif ou abstrait, est devenu nécessaire en raison de la quantité exponentielle de contenu textuel et de diverses archives d'actualités, d'articles scientifiques, de documents juridiques, etc.

Ce thème propose un résumeur abstrait du texte arabe à l'aide du base de données XLSum, en appliquant des méthodes d'apprentissage en profondeur et en utilisant des modèles pré-entraînés. Nous proposons deux systèmes de résumé. Le premier repose sur des approches d'apprentissage profond. Nous utilisons un modèle pré-entraîné AraBART de la langue Arabe et l'ajustons pour la tâche de résumé. Le second est l'application du PEGASUS, le résumeur anglais, au texte Arabe. De plus, nous évaluons leur efficacité.

Mots clés: Résumé automatique de texte, Extractif, Abstrait, Langue Arabe, TLN, Apprentissage en profondeur, Plongement de mots, BERT, BART, PEGASUS.

ملخص

أصبح تلخيص النص، الذي يكون عادة استراتيجي أو تجريدي، ضروريا نظرا للكم الهائل من المحتوى النصي والأرشيفات المختلفة للأخبار والأوراق العلمية والوثائق القانونية وما الى ذلك.

يقترح هذا الموضوع تلخيصا تجريديا للنص العربي باستخدام مجموعة بيانات XLSum، من خلال تطبيق أساليب التعلم العميق، واستخدام النماذج المدربة مسبقا. نحن نقدم نظامين للتلخيص. أولا نستخدم نموذج AraBART المدرب مسبقا على اللغة العربية ونقوم بضبطه لمهمة التلخيص والذي بدوره يعتمد على مناهج التعلم العميق. والثاني هو تطبيق PEGASUS، الملخص الإنجليزي على النص العربي. بالإضافة الى ذلك، نقوم بتقييم فعاليتهم.

الكلمات المفتاحية: تلخيص تلقائي للنص، استخراجي، تجريدي، اللغة العربية، معالجة اللغة الطبيعية، التعلم العميق، تضمين الكلمات، BERT، BART، PEGASUS.

Summary

List of Figures

List of Tables

General introduction

In the face of the advent of the Internet and search engines, textual information in electronic form accumulates quickly and in very large quantities. Therefore, it is interesting to offer computer tools for quick visualization of texts, such as automatic summarizers, so the user can assess how well the document fulfils his need. To summarize is to provide a brief presentation that includes the basics of the source text. Its purpose is to help the reader identify information that may interest him without having to read the entire document and by means of which he determines whether he wants to complete the reading of the whole document or not. The task of automatic summarization is attributed to the field of natural language processing, as the first thing that was considered was at the end of the fifties, specifically in 1958 [1]. Then it was developed and various systems related to the English language were established, and state-of-the-art results were reached.

It has been observed that the automatic summary is based on two main approaches, extract and abstract. Extractive when the summary is the important sentences extracted as they are from the source without any change [2]. Abstractive is a convey of the most important information contained in the text in another form [2]. Although the abstract is better understood and readable, most work in the field of summarization relies on extraction, which usually lacks consistency.

Despite the passage of years from the first appearance of this field and the developments reached by researchers in the summarization, the Arabic language has not received much work on it. The most popular works were of widely used languages and their relatively simple grammar, such as English. Now there is a need for automatically summarizing Arabic texts. Because it is the official language in 25 countries, it is spoken by more than 460 million people in the world, and the number is still increasing. The reasons for the lack of research in it are the difficulties of processing the complex natural Arabic language in terms of structure and morphology, as well as the lack of Arabic resources and the absence of the gold standard summary [3]. Our theme is focused on creating an abstractive text summarization system for texts written in Arabic.

Problematic and motivation

The Arabic language is one of the most complicated languages in structure and morphology, which made the progress of research in the field of natural language processing slow in it. However, the massive amount of data available recently made the availability of automatic summarizers supporting this language an urgent need.

Nowadays, pre-trained models are widely used for NLP tasks and can be used to summarize an Arabic document.

Aim of the work

The objective of this study is to provide a summarizer based on pre-trained models to convey the most important information in an abstract manner from the source text.
The following main objectives can be identified:

- Identifying the domain and its components.
- Fine-tuning of pre-trained models to perform the task of Arabic summarization.
- Comparisons of the obtained results.

Dissertation structure

The Dissertation is organized as follows:

- **Chapter 1 Automatic text summarization:** This chapter paves to the field of natural language processing and discusses an outline of key aspects of text summarization and some related works.
- **Chapter 2 Word embedding for text summarization:** This chapter provides the core of our theme which includes word embedding, as well as a deep learning overview with mentions of models BERT, BART and PEGASUS.
- **Chapter 3 System design:** This chapter describes the design architecture, including all processes involved in building the abstractive Arabic text summarizer.
- **Chapter 4 Implementation and results:** This chapter introduces the tools for implementing the solution, explains the code, and discusses the final result.
- **Conclusion and future work:** This chapter summarises the most important findings and makes recommendations for further research.

Chapter 1

Automatic text summarization

1.1 Introduction

Over the past two decades, the need to automatically summarize documents has seemed to increase due to the large mass of online texts such as news, scholarly articles and so on. Assigning the summarization task to a machine has become an urgent need to extract and generate key information from the vast amount of text available. Text summarization has been an area of intense research for the past 50 years, particularly for commonly used languages and relatively simple grammar such as English. However, the Arabic language did not receive much attention due to its complexity grammatically and morphologically etc. Arabic presents researchers and developers of Natural Language Process (NLP) applications a significant challenges due to its difficulties.

It is necessary to evaluate the effectiveness of the summaries generated by the machine to determine the extent of its reliability. There are two methods, either human evaluation, that is, manual evaluation, which is not a practical method in the case of large data sets, or automatically using the ROUGE metric, which is the one that we adopt in this study.

In this part, we try to define the subject of our study, which is text summarization. So, we begin first with some definitions and applications of natural language processing (NLP). We next outline the process of summarization in general, including the way in which it is done manually or automatically and the different types and aspects of summaries. Then we'll go into the Arabic characteristics, as well as how this language was encoded and its challenges. Finally, we present some recent work in this field and finish with the ROUGE evaluation metric.

1.2 NLP

1.2.1 Problem definition

We all know that machines cannot understand human languages such as speeches, texts, or even letters, so converting these texts into machine-understandable format has become a crucial goal to enable them to perform useful tasks. The domain of study that concentrates on enabling computers to intelligently process human languages is termed **N**atural **L**anguage **P**rocessing or **NLP** for short. It is a multi-speciality field that combines artificial intelligence, computing science, cognitive science, information processing, and linguistics. Natural language processing appeared to facilitate the work of the user and satisfy the desire to communicate with the computer in a natural language.

NLP is divided into two parts: Natural Language Understanding and Natural Language Generation. **N**atural **L**anguage **U**nderstanding (**NLU**) is the part of natural language processing that helps computers understand and interpret natural language as a human being do. It is used for machine translation, question answering and many others. **N**atural **L**anguage **G**eneration (**NLG**) is a component of natural language processing, which is the process of creating meaningful text in the form of a natural language from an internal representation. It can be considered the opposite of natural language understanding [4]. NLG systems offer an important role in text summarization, machine translation, and dialogue systems.

1.2.2 Applications of NLP

Perhaps with the exception of AI researchers, NLP is not a purpose in itself but rather a tool for performing a certain task. The most common applications that use NLP include:

- **Machine Translation (MT):** Natural language processing is at the core of machine translation as it translates natural languages automatically (from one language to another) using machines. The machine translation challenge is not translating word by word but maintaining the true meaning of sentences with grammar and tenses [4].
- **Question answering (QA):** Question answering is one of the top tasks in Natural Language Processing (NLP), in which the system takes a natural language question as input and returns either the text answer or answer-providing passages.
- **Sentiment Analysis (SA):** The field of sentiment analysis or opinion mining has received a lot of focus from NLP. It is the field of analyzing people's opinions, sentiments, evaluations, and reviews to determine the positive or negative orientation of user-generated texts about entities such as products, services, and others [5].

- **Summarization:** It is the process of conveying the most important information from a source (or sources) to produce a condensed version. This task took advantage of the NLP to reformulate the generated summary into a more coherent text [6].

There are many other applications of NLP. It should be said that every application that uses text is a candidate for NLP, such as Information Retrieval (IR), Text Classification, Text Generation, Sentence Similarity and so on. We will continue to expand on the task of automatic text summarization as long as we are interested in it.

1.3 Text summarization

After intensive research on the meaning of summary, I found many meanings and definitions. Here are some basic definitions from different sources. In the Cambridge Dictionary [7], "a summary is a short, clear description that gives the main facts or ideas about something". Karen Sparck Jones [8] takes a definition of a summary as "a reductive transformation of source text to summary text through content reduction by selection and/or generalisation on what is important in the source". As stated by Dragomir Radev et al. [9], a summary can be loosely defined as a "text that is produced from one or more texts, that conveys important information in the original text(s), and that is no longer than half of the original text(s) and usually significantly less than that. Text here is used rather loosely and can refer to speech, multimedia documents, hypertext, etc.".
So, in general, we can say that to summarize something is to give a brief presentation, which includes the basics of the text. Its purpose is to help the reader identify information that may be of interest to him without having to read the entire document.

There are two manners in which the summary is created. We distinguish two types: manual done by human and automatic done by machine.

1.3.1 Manual summarization

Making a summary requires a deep analysis of the textual content and intellectual abilities in order to understand the content, take what is important and leave everything else. Producing a summary is therefore a complex task. In fact, for humans to prepare a summary of any kind of thing (texts, films, events, etc.), understanding is necessary and integral to it.

According to [10], the process that a person follows to produce a summary (Figure 1.1) consists of five steps. These steps are: full reading of source text, analyzing of source information, prioritizing the information, synthesizing the information and writing the summary text. We'll go over these processes in more detail below. [10].

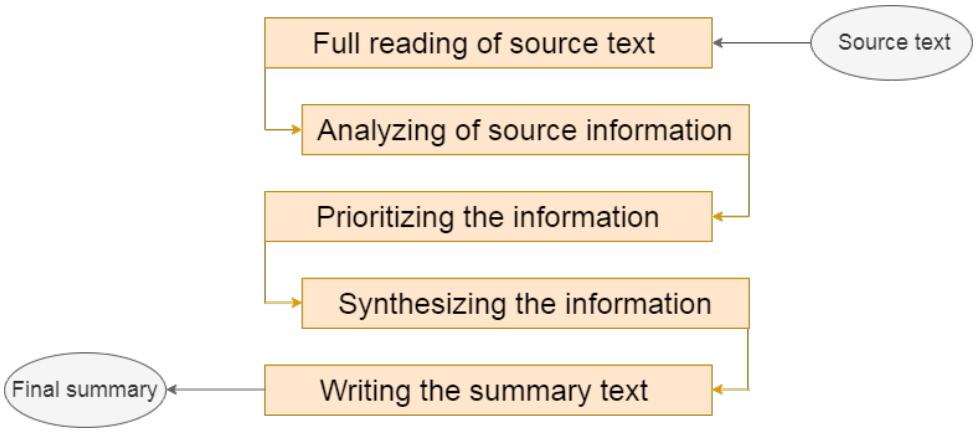

Figure 1.1: Human process for summary by comprehension

First, the full reading of the source text, the person who wants to write the summary should read the full text carefully one or several times to be informed about it, understand its vocabulary, and be sure about the content and meaning. Also, he observes the layout of the text: its titles, paragraphs, indents, etc. Second, analyzing source information, the reader analyzes only the information present in the source text such as specific passages, keywords, linking words, connectors, etc. He discusses with himself the meaning of each expanded notion while remaining neutral and objective. This comprehension allows the author to know how to distinguish the most important information that should appear in the summary. Third, prioritizing the information, the writer sorts the information contained in the text, as he categorizes the main ideas and information for a good understanding of the text and abandons semantic links. Fourth, synthesizing the information, selected information should be reduced and assembled. This reduction can take place using processes aimed at reducing the number of words and phrases. Finally, writing the summary text. It is rewriting text that is shorter than the initial text with a neutral style. The synthesized information from the source text is committed to be mentioned without any comments, illustrations, etc.

1.3.2 Automatic summarization

Mani Maybury [11] stated that summarization is "the process of distilling the most important information from a source (or sources) to produce an abridged version for a particular user (or users) and task (or tasks)". When this is achieved through a computer machine, i.e. Automatically, we name this Automatic Text Summarization [12]. It is "an automatic technique to generate a condensed version of the original documents" [13]. This idea, producing summaries automatically, is not new, its origin dates back to the mid 20th century, precisely in 1958 which was first discussed openly by Luhn [1].

As long as we are interested in automatic summarization, we will expand on it. We are going to talk about its types, in addition to the main approaches used. And since our research concerns the automatic summarization of the Arabic language, we will discuss its challenges and problems that hinder it.

1.4 Automatic Text Summarization

Automatic text summarization is a type of natural language processing in which a machine is given a task to analyze, comprehend, and use human language to generate a summary. The type of summary results is determined by a number of aspects, the most well-known of which are extractive and abstractive, both of which have advantages and drawbacks.

1.4.1 Automatic text summarization aspects

The summary result is based on three main aspects which are output, input, and purpose. Figure 1.2 depicts the most essential ones.

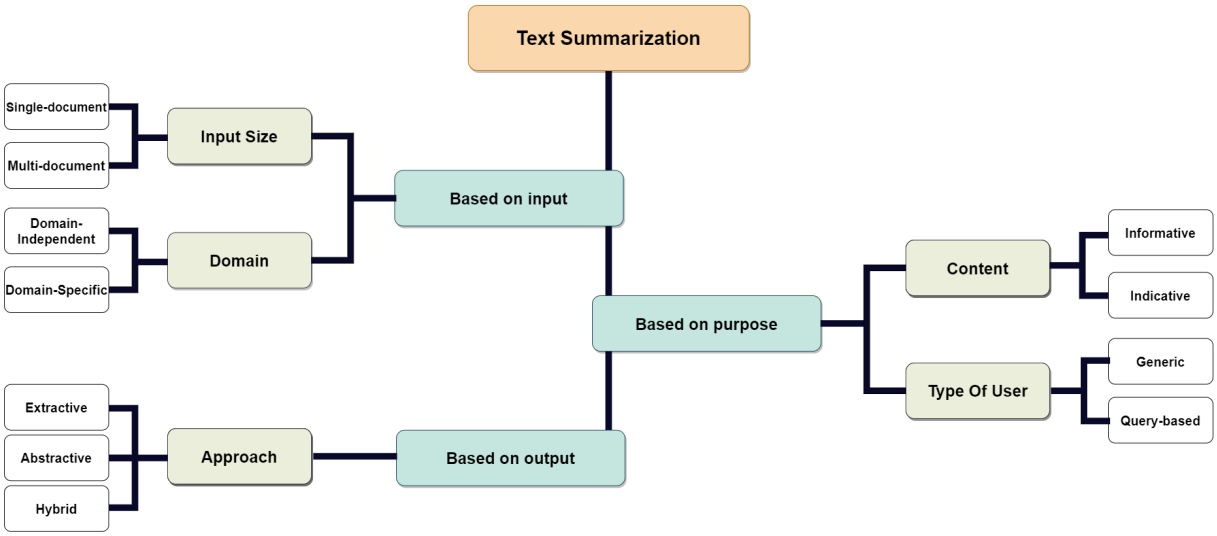

Figure 1.2: Automatic text summarization aspects

- **Based on approach:** Wafaa S El-Kassas et al. [14] considered that text summarization can be Extract, Abstract or Hybrid. An extract is a collection of the most important portions, extracted from the source verbatim without modifying the selected text [2]. In contrast, the abstract summary simulates what a human does. It is a rewrite and paraphrase of the source text information in a concise and understandable way. The hybrid text summarization approach here merges both the extractive and abstractive approaches.

- **Based on content:** Here the focus is on the content of the summary result, it can be informative or indicative. The informative summarization contains all relevant information and also reports all main topics [14]. However, the indicative summarization takes only the themes developed in the source text, for the reader can judge whether he should consult the source document or not [14].

- **Based on input size:** We distinguish two types of input: Single-document or Multi-document. In the case of Single-document, the system processes only one document at a time. On order to make the document shorter without losing important information [15]. In the other hand, Multi-document, the system accept a collection of documents to generate one summary. And the point is to eliminate recurrent content in the set of input documents [15].

- **Based on language:** Summaries are divided into 3 groups depending on the language of the input text and the output summary [16]. They are Monolingual, Multilingual, or Cross-Lingual. The first is when the source and the resulting summary have the same language. The second is when the system processes documents in different languages and the summary is generated in these languages. And the third is when the source text is in a language and the generated summary is in a different language.

- **Based on the type of user:** Text summaries can be categorized according to the audience. It can be generic or query-based summaries. About the generic one, generally, in this type of summary, all relevant features of a source text are represented and all main topics are given the same level of important [17]. For the reason that it is assumed that the summary may be used by different types of users. In other words, it does not take into account what the user wants to search for. In contrast, in query-based summarization, only the information in the original text that is relevant to a specific user query is summarized.

- **Based on the domain:** Concerning the domain, we distinguish two types: Domain-Independent and Domain-Specific. The Domain-Independent system is not restricted to certain domains, it summarizes any document genre and can accept different types of texts. For a domain-specific system, only documents in a specific domain are accepted, i.e. the summary is generated from a text entry in a single restricted domain [18].

- **Based on summarization algorithm:** The task of summarization can be either supervised or unsupervised. In a supervised algorithm, a training phase is necessitated on considerable quantities of training data to determine the important content from the input documents [16]. This training data is labelled by humans manually. In contrast, in the unsupervised algorithm there is no need for the training phase or training data [14]. It uses unlabeled data to generate summaries from the input documents.

1.4.2 Automatic text summarization approaches

Automatic Text Summarization can be done in a variety of ways, but we will focus on two of them: extraction-based and abstraction-based summarization.

1.4.2.1 Extractive approach

An extractive summary [19] consists of relevant sentences selected from documents that cover the main contents without any modification on them. These sentences are selected based on statistical and linguistical features. According to these features, a score of importance is given to each sentence. Only those with higher scores will be chosen to represent the final summary and will be organized in the same order they were in the original text [20]. The facility of implementing such a process is that it does not need deep text understanding [21, 22].

Advantages of extractive approach

The extractive approach may give an effective and efficient summary [23]. This type of summarization is easy to compute because it does not handle semantics [24]. This technique extracts sentences directly, thus, scores a higher accuracy [25]. Besides, the gist is conveyed to the summary with the same terms used in the source text. So, the reader does not have to worry about the misinterpretation of the text [25].

Disadvantages of extractive approach

The extracted sentences are often long, which makes it more likely that irrelevant information will also be included in the summary, which consumes space [6, 22]. Extractive summaries cannot capture all important information because it is often disseminated via sentences [22]. Therefore, not much accurate information was provided [21]. This extraction often leads to higher incoherence in the summary sentences [23], endures from inconsistencies and lack of balance [24]. Since this technique extracts the sentences as they are, there is a high possibility that the pronouns lose their references [26]. In addition, these types of summaries are less readablees [27].

1.4.2.2 Abstractive approach

Abstractive summarization is paraphrasing the source text information concisely and understandably, as a human does. This is done with the help of a linguistic method for understanding and examining the original text [28]. Therefore, it needs extensive natural language processing to have the ability to rewrite and paraphrase important information from text documents [16]. In this approach, an internal semantic representation is built, and then the summary is generated using natural language generation techniques [29].

Advantages of abstractive approach

Since the abstract method necessitates a deep analysis of the text and the capability to produce new sentences, its generated results are closer to the manual summary [30]. This has several benefits, including reducing redundancy [31, 32], maintaining a good compression rate [24, 31, 32], being more concise [24], semantically relevant [24] and condensed more strongly than extractive ones [33].

Disadvantages of abstractive approach

The first issue that faces this approach is the difficulty to generate such a type [23]. And its difficulty is followed by the hardness of developing a program that allows it to be done [33]. Representation is also considered the biggest problem facing abstractive summaries [21, 22]. The richness of representations of abstractive systems controls their capabilities since they cannot summarize what their representations cannot capture [21]. Another drawback, according to [23], current abstractive systems often suffer from the matter that they make repetitive words, also not treating words out-of-vocabulary appropriately.

1.5 Arabic language and its challenges

There is no doubt that Arabic is an interesting and difficult language at the same time, but it attracts many researchers to be their subject and many foreigners to learn it. It is interesting because of the importance of its cultural and literary heritage. In addition to the fact that there are more than 460 million speakers of Arabic in the world, and it is an official language in the United Nations and 24 other countries. As a result, we will go over the language in greater depth, as well as some of the challenges that may arise.

1.5.1 Arabic language

Before getting into the depth of the Arabic language, we will first introduce the language.

Arabic is the most widely spoken Semitic language, and it is the native spoken medium of communication for everyday life. However, the term "Arabic language" is frequently used to refer to Modern Standard Arabic (MSA) and its dialects as a whole. The media and education use MSA as their primary written language [34]. The Arabic is written from right to left using the 28-character Arabic alphabet. It is written in a cursive way, and the shape of each letter changes according to its position in the word beginning, middle, or end, but it does not contain capital letters. As for vowels, it can be said that they are two types: long letters and diacritics.

Despite the fact that the Arab language-dialect situation appears to be similar to that of many other languages around the world, it differs in two ways [34]. Firstly, a great disparity between standard Arabic and its dialects, which are frequently compared to Latin and Romance languages, such as Algerian Arabic has a lot of influences from Berber as well as French, where Arabic dialects substantially differ from MSA, and each other in terms of phonology, morphology, lexical choice, and syntax [35]. Secondly, the standard Arabic language is not the native tongue of any Arab [36].

In the Arabic script, there are two sorts of symbols for writing words: letters and diacritics, as well as additional symbols like digits and punctuation.

There are 19 distinct letter shapes as shown in figure 1.3

Figure 1.3: Letter forms [35].

In addition to three types of letter mark: dotes, short kaf, and Hamza.

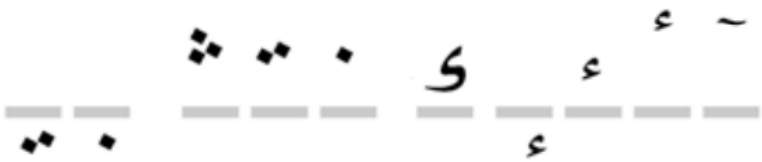

Figure 1.4: Letter marks [35].

For the diacritics, Written Arabic can have complete diacritics, partial diacritics, or no diacritics at all. We can find diacritized text in religious texts, children educational texts, and some poetry. As well as in modern written Arabic, several diacritics are used to assist readers distinguish between different words. There are types of diacritics that are Vowel, Nunation, and Shadda [35].

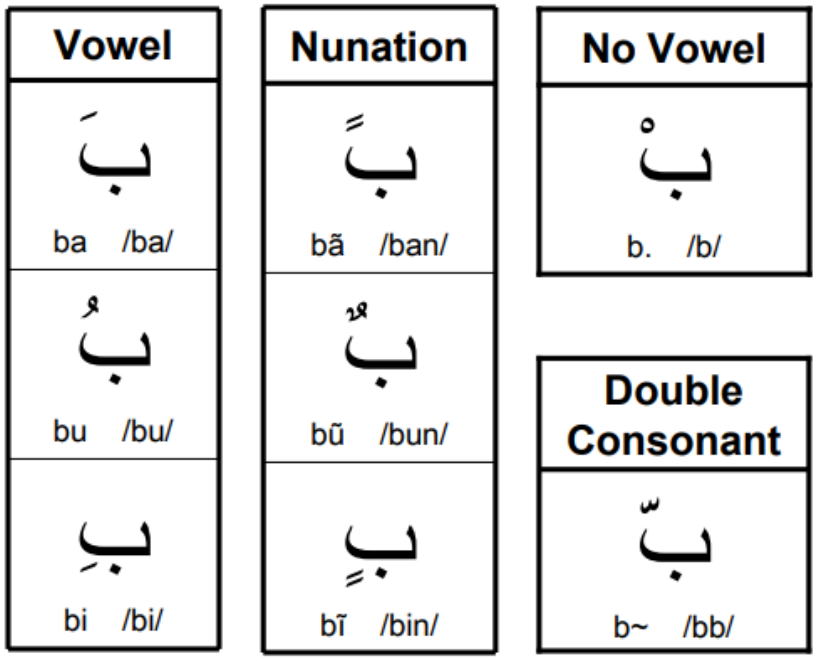

Figure 1.5: Types of Arabic diacritics [35].

There are 36 letters in Modern Standard Arabic (MSA), which may be divided into the following subsets [35]:

- **The basic 28 letters:** The Arabic alphabet's core 28 letters correspond to the 28 consonantal sounds in Arabic. Except for the Hamza letter form, they are made up of all the letter forms. The letter markings in all of these letters are fully discriminative, allowing you to discern between distinct sounds.

- **The Hamza letters:** There are six of them, the original one is the "Hamza-on-the-Line", which is made up of the Hamza letter form (ء). The others employ different letter forms with the Hamza and Madda letter marks (أ إ ئ آ ؤ).

- **The Ta-Marbuta:** This letter is a unique morphological identifier that denotes a feminine ending. The Ta-Marbuta (ة) is a hybrid letter that combines the Ha (ه) and Ta forms (ت).

  Ta-Marbuta only appears at the end of words. The letter Ta (ت) is used when the morpheme it represents is in word-medial position.

- **The Alif-Maqsura:** This letter also serves as a unique morphological identifier, indicating everything from feminine ends to underlying word roots. The Alif-Maqsura

(ى) is a hybrid letter that combines the Alif (ا) and Ya (ي) forms. The Alif-Maqsura only appears as a dotless Ya at word-final places. The letters Alif or Ya are used when the morpheme it represents is in word-medial position.

For the digits, in the Arab world, there are two sets of digits are utilized to write numerals. Only Western Arabic nations employ the Arabic numerals that are generally used in Europe, the Americas, and the rest of the globe (Morocco, Algeria, and Tunisia). Among the Middle Eastern Arab countries that employ Indo-Arabic numbers are Egypt, Syria, Iraq, and Saudi Arabia. Despite the fact that Arabic is written from right to left, the multi-digit number forms are identical to those used in European (left-to-right) languages [35].

Figure 1.6: Arabic digits [35].

The Tatweel or kashida (ـ) is a particularly unique Arabic symbol. The Tatweel is a word-stretching symbol that is used to emphasize words or simply to force vertical justification. Tatweel serves a comparable purpose as utilizing capital letters for prominence or emphasis in Arabic because there is no Arabic counterpart to capital letters. Tatweel is nearly often removed as part of the preparation of Arabic text for NLP applications in order to decrease sparsity.

1.5.2 Arabic encoding

An encoding is a method of representing the symbols in a script in a consistent manner for storage and access by machines. When it comes to encoding design and how it interacts with data storage and access, the Arabic alphabet has some unique issues. This is mostly due to the differences between Arabic and European scripts.

Each complex ligature and letter form with distinct diacritics can be represented as a complicated "symbol" in an encoding. The encoding's number of distinct symbols grows rapidly. Different letter markings, on the other hand, can be encoded as distinct symbols from letter shapes and diacritics. Unicode, CP-1256, and ISO-8859 are the most widely used encodings schemes of Arabic.

The International Standards Organization (ISO) established ISO-8859-6, which is similar to the Arab Standards and Metrology Organization's ASMO-708 standard (ASMO). Microsoft created CP-1256, sometimes known as Arabic Windows encoding, and it quickly became popular. Each symbol is represented by one byte (8 bits) in both of these encodings. The first 7 bits are reserved for English ASCII, as they are in other encodings in

their class for scripts/languages other than Arabic. The remaining 128 characters reflect the other script.
Currently, unicode has become the de facto standard for concurrently encoding a vast variety of languages and scripts. Unicode was created with only two bytes of data in mind, but it has now grown to include over one million different symbols. Unicode has a large number of Arabic characters. It also assigns unique addresses to Arabic letter forms and ligatures [35].

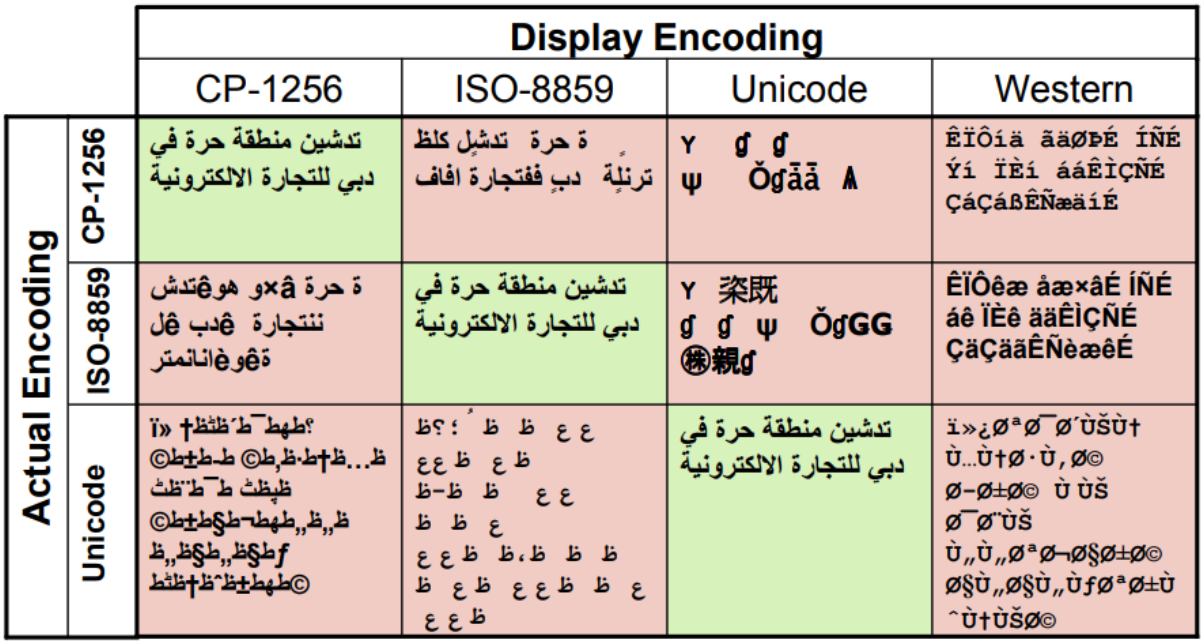

Figure 1.7: Comparing the decoding of various Arabic encodings [35].

1.5.3 Arabic language challenges

The complexity of the Arabic language complicates the various **A**rabic **N**atural **L**anguage **P**rocessing (**ANLP**) tasks such as word derivation, machine translation and automatic summarization [2]. This causes many challenges, the most important of which are:

- As we mentioned earlier, changing the writing of letters depends on their position in the word.

- Diacritics are optional and not necessary, as they are usually absent in texts and articles, which requires sophisticated analysis to know the correct meaning of the word and sentence [37].

- Arabic is highly derivational and inflectional [2, 38, 3], It is possible to reach three and more adding letters in many forms.

- As the Arabic language is broad, it contains many words with the same meaning and different meanings for the same word, which increases the degree of its ambiguity.
- All of the aforementioned affected the interaction of researchers with this language and resulted in a lack of natural language processing tools and resources (corpora, dictionaries, lexicon, etc.) [38].

These challenges make it inappropriate to summarize the Arabic text using the methods used to summarize other languages, such as English [3].

1.6 Related works

We will go through some of the previous work that employed natural language processing techniques for text summarization.

1.6.1 Before the emergence of BERT

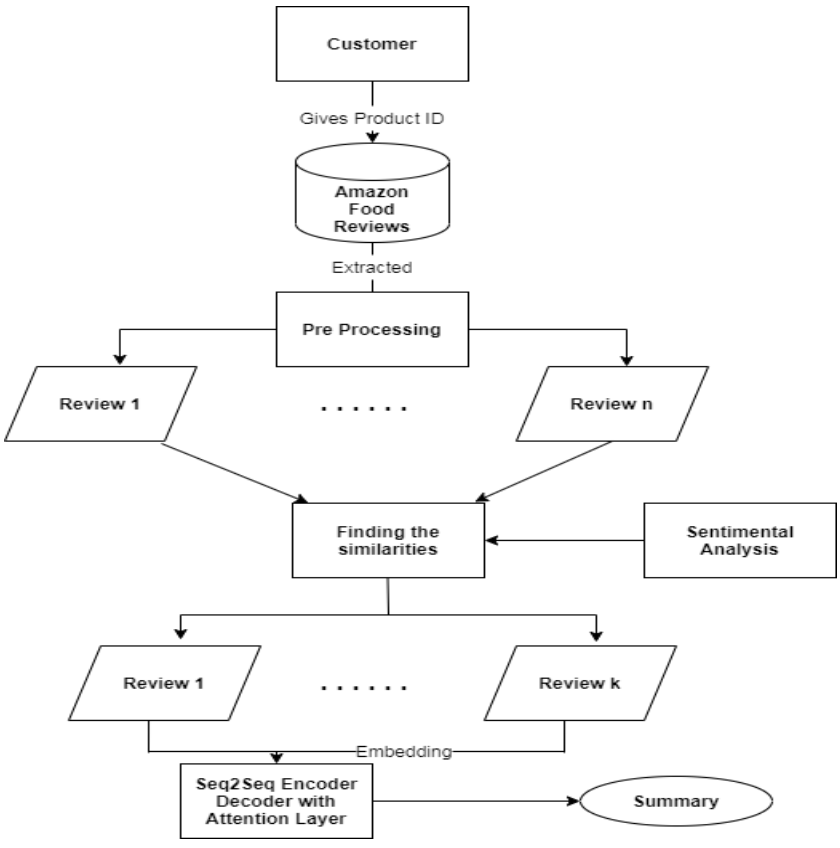

Figure 1.8: System architecture proposed by Janice Shah et al. [39].

Janice Shah et al. [39] proposed a solution to this issue using machine learning and natural language processing represented by its architecture in Figure 1.8. They used an

abstraction based summarization and Sentimental Analysis for in-depth learning about reviews. They built a system that was applied to English Amazon food reviews. It takes as input the id of the product that the user wants to summarize its reviews. After pre-processing the reviews, The IBM Watson tone analyzer is used to determine the sentiment of these reviews. A universal encoder used by Google subsequently is used to detect similar reviews in order to delete them. Then, the remaining ones are embedded using Conceptnet-numberbatch embedding and given to the seq2seq encoder-decoder to obtain the review's summary.

In this work [40], an extractive statistical system was proposed to summarize the Arabic single and multi-document language, following the steps shown in the Figure 1.9. After pre-processing the text to be summarized, similar sentences are clustered, and then mRMR (minimum redundancy and maximum relevance more information on it can be found in this article [41]) is applied to them to find the highly discriminant terms and thus choose the most appropriate sentences to represent the text content. In addition, a new algorithm has been proposed to extract sentences with high-ranking terms and maximum diversity.

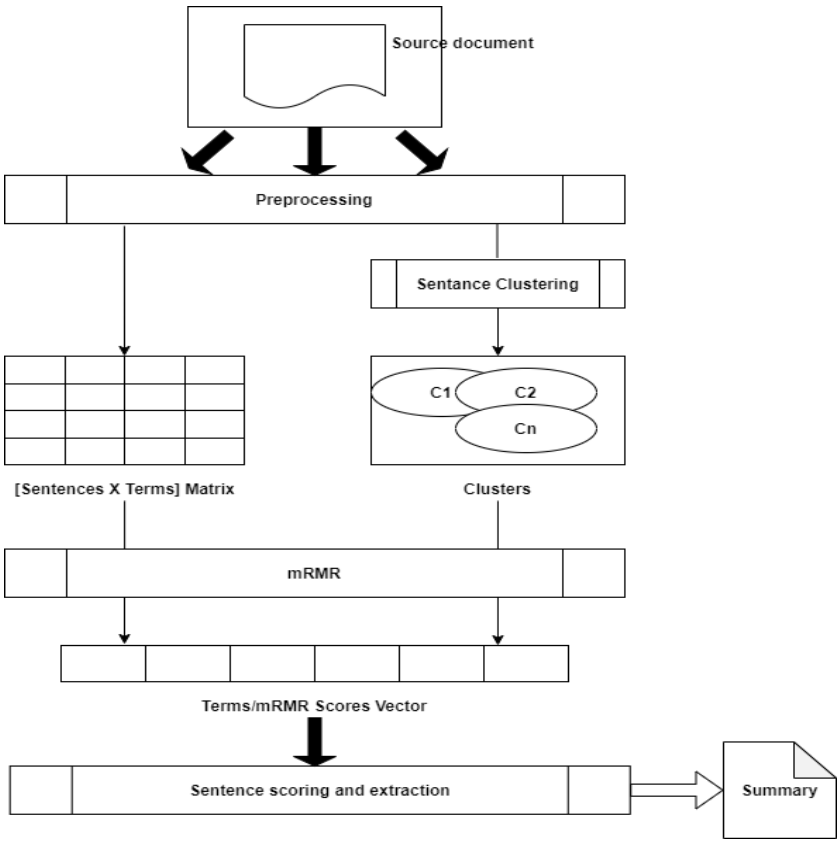

Figure 1.9: System architecture proposed by [40].

A hybrid approach to summarizing the single-document Arabic text is presented in this work [42]. It included the use of domain knowledge, statistical features and genetic

algorithms to select and extract the most important points contained in the political document written in Arabic. First, a preprocessing of the document is performed, such as sentence segmentation and tokenization and extraction of domain keywords. Then, a score is assigned to each sentence in the document to determine its importance based on its features such as the presence of domain-specific keywords in the sentence, and the similarity of the sentence to the document title. After this stage, the genetic algorithm is applied to produce a readable and coherent summary based on the cohesion between sentences and similar to the subject of the document. Figure 1.10 presents The architecture of this work.

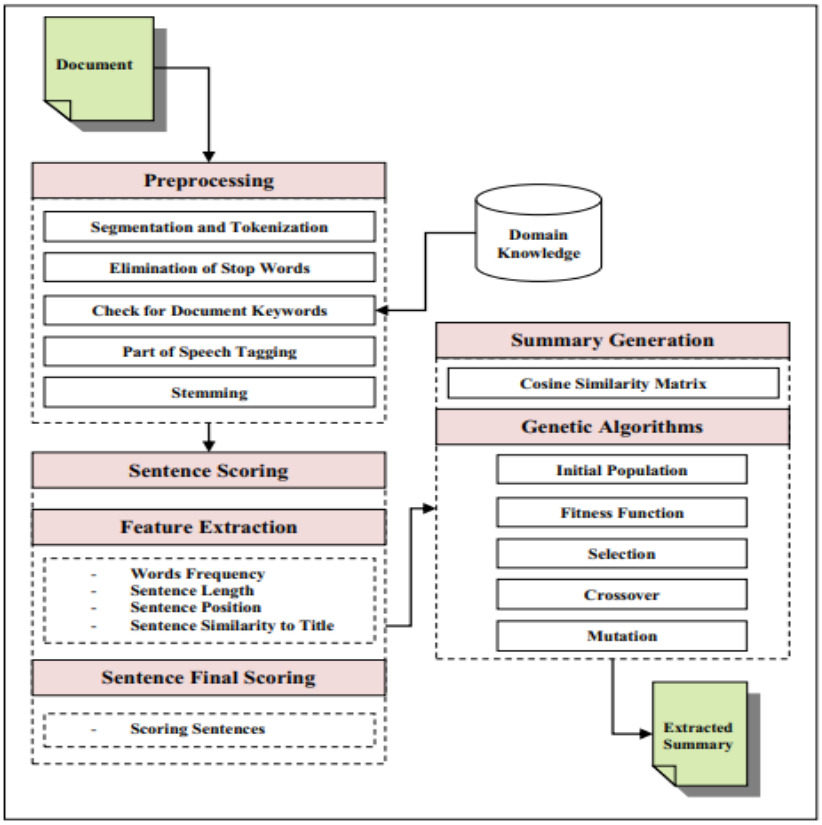

Figure 1.10: System architecture proposed by [42].

1.6.2 Usage of BERT after its emergence

Abdullah M Abu Nada et al. [43] proposed a single document extractive Arabic text summarization by combining NLU using the (AraBERT) model and clustering technique. After preparing their input, they used the AraBERT pre-trained model to generate an embeddings matrix. Next, the K-Means model was chosen for the clustering, and from its results, the nearest sentences to each cluster centroid were selected as a summary. Some of the weaknesses of this system include a decrease in the coverage accuracy when the text is too long, and sometimes the extracted sentences contain a linguistic expression that causes misunderstanding of the summary.

Another work presented based on DistilBERT (distilled version of BERT) is called ArDBertSum for summarizing Arabic texts [44]. Before executing DistilBERT, two distinct text preprocessing procedures is performed as shown in Figure 1.11. It applies 3 preprocessing tasks which are normalization, noise removal, and sentence segmentation. It is intended to ignore tasks that break the sentence structure, like removing stop words, in order to produce a readable candidate summary after pre-processing. Then, DistilBERT is used to consolidate disparate textual representations: represent the input document embeddings, tokenize it into sentences where each sentence begins with [CLS] and ends with [SEP], and choose which sentences to create as a candidate summary.

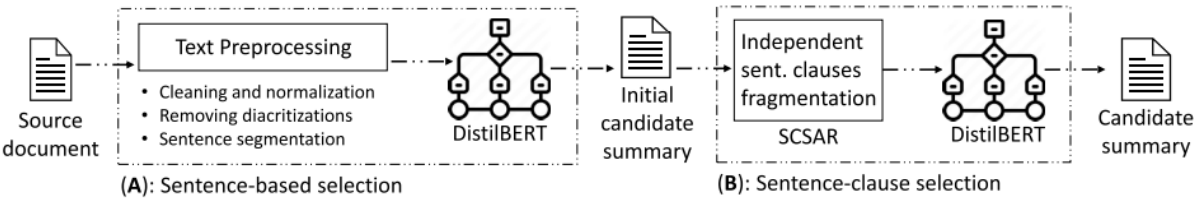

Figure 1.11: The proposed DistilBERT-based method by [44].

1.7 Evaluation of summaries

ROUGE, stands for **R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation, which is the most widely used method for evaluating automatic text summarization. It is a software framework that includes several metrics to measure the similarity between automatic summaries and perfect human-generated summaries (Golden) by measuring, for example, the number of overlapping units such as n-gram, word sequences, and word pairs [45]. The most common varieties are ROUGE-N (N-gram Co-Occurrence Statistics), ROUGE-L (Longest Common Subsequence), ROUGE-S (Skip-Bigram Co-Occurrence Statistics), and ROUGE-SU (Extension of ROUGE-S).

- **ROUGE-N: N-gram Co-Occurrence Statistics** is an n-gram recall between two summaries (generated and reference) to count the number of matches between them, where N refers to the length of the N-gram. For example, ROUGE-1 refers to overlapping unigrams between the two summaries, and ROUGE-2 refers to the overlap of bigrams between them. The formula that calculates ROUGE-N is [45]:

$$ROUGE - N = \frac{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} Count(gram_n)} \tag{1.1}$$

  Where:

- n: stands for the length of the n-gram.
- $Count_{match}$ *and* $gram_n$: is the maximum number of n-grams co-occurring in a candidate summary and a set of reference summaries.
- *Count* is the number of N-grams in the set of reference summaries.

- **ROUGE-L: Longest Common Subsequence** calculates the ratio between the longest common subsequence (LCS) of the generated summary and the length of the reference summary [46]. One of LCS's requirements is in-sequence matches that reflect sentence-level word order as n-grams. One of its advantages is that it does not require consecutive matches and does not have to predefine the length of n-grams. Its second advantage is that it automatically includes the longest in-sequence common n-grams [45]. The formula that calculates ROUGE-L is [45]:

$$R_{lcs} = \frac{LCS(X,Y)}{m} \tag{1.2}$$

$$P_{lcs} = \frac{LCS(X,Y)}{n} \tag{1.3}$$

$$F_{lcs} = \frac{(1+\beta^2)R_{lcs}P_{lcs}}{R_{lcs} + \beta^2 P_{lcs}} \tag{1.4}$$

Where:

- X is a reference summary sentence.
- Y is a candidate summary sentence.
- m is the length of the reference summary.
- n is the length of the candidate summary.
- *LCS(X,Y)*: of a longest common subsequence of X and Y.
- beta is the relative importance of P_{LCS} and R_{LCS} which equals:

$$\beta = \frac{P_{lcs}}{R_{lcs}} \tag{1.5}$$

- **ROUGE-S: Skip-Bigram Co-Occurrence Statistics** measures the interference ratio of skip-bigrams between a candidate summary and a set of reference summaries, where a skip-bigram is any pair of words in a sentence in order, allowing for arbitrary gaps. For example, for the phrase "*cat in the hat*" the skip-bigrams would be "*cat in, cat the, cat hat, in the, in hat, the hat*". The formula that calculates ROUGE-L is [45]:

$$R_{skip2} = \frac{SKIP2(X,Y)}{C(m,2)} \tag{1.6}$$

$$P_{skip2} = \frac{SKIP2(X,Y)}{C(n,2)} \tag{1.7}$$

$$F_{skip2} = \frac{(1+\beta^2)R_{skip2}P_{skip2}}{R_{skip2} + \beta^2 P_{skip2}} \tag{1.8}$$

Where:

- *SKIP2(X,Y)*: is the number of skip-bigram matches between X and Y.
- C: is the combination function.
- beta is the relative importance of P_{skip2} and R_{skip2} which equals:

$$\beta = \frac{P_{skip2}}{R_{skip2}} \tag{1.9}$$

- **ROUGE-SU (Extension of ROUGE-S):** is defined as a weighted average between ROUGE-S and ROUGE-1 [46]. ROUGE-S considers only bigrams which gives a score of zero if the candidate summary sentence is the exact reverse of the reference summary sentence, so there is no skip bigram match between them. Due to this weakness, ROUGE-SU comes with the use of unigram with bigrams.

1.8 Conclusion

In the current chapter, we went through the most important terminology involved in our theme, which is based on automatic text summarization. We started with the concept of natural language processing and its applications, and then we delved deeper into our topic which includes text summarization. We also learned about the Arabic language and its challenges. Furthermore, we provided a wide overview of our chosen topic by providing similar work on the subject alongside each approach.

In the following chapter, we will discuss deep learning-based word embedding and a detailed explanation of the approach that we will use.

Chapter 2

Word embedding for text summarization

2.1 Introduction

When building models whose aim is to grasp and interpret natural languages such as human languages, it is not practical for these models to directly interact with text data. They rely mainly on statistical, mathematical, and optimization techniques. They are only able to understand numbers. This necessitates preprocessing text data and transforming it into a representation of matching scalar, allowing models to make computations and decisions. Embedding algorithms efficiently find a corresponding numerical representation of text words.

Word embedding is the collective name for a group of language modelling and feature learning techniques in NLP in which words or phrases from vocabulary are mapped to vectors of real numbers. With the increasing human need for more effective representations that express their language fluently, embedding algorithms have evolved from representing the word with a number to represent it with vectors, and from having only one representation for each word, it has become possible for one word to have multiple representations depending on the context in which it appears. Here algorithms are divided into three types, frequency-based, static and contextual. To perform the task of summarization, these types of word embedding are used. What interests us in this work is the contextual embedding algorithms based on deep learning that help in providing a suitable representation of the intended meaning of the words in the text. The most famous development is BERT and its derivatives, which we will address later in this chapter.

This chapter seeks to introduce the concept and basic types of word embedding algorithms. In addition to getting a general understanding of deep learning and the attention mechanism that is the building block of transformers, we will get to know the latter as well. Our chapter comes to a close with an overview of BERT, BART and PEGASUS.

2.2 Word embedding

Word embedding, also known as distributed word representation, is the representation of words with a set of real numbers in the form of a vector for example, to capture the semantic and syntactic information of words from the unlabeled text data set [47]. This representation enables us to relate words together that appear from a similar context where words of similar meaning have a similar representation. They regard the process of word embeddings as scattered representations of text in n-dimensional space that attempt to capture the meanings of the words [48].

Traditional word embedding (like TF-IDF), Static word embedding (like Word2Vector), and Contextualized word embedding (like BERT) are the three most common word embeddings types, according to [48].

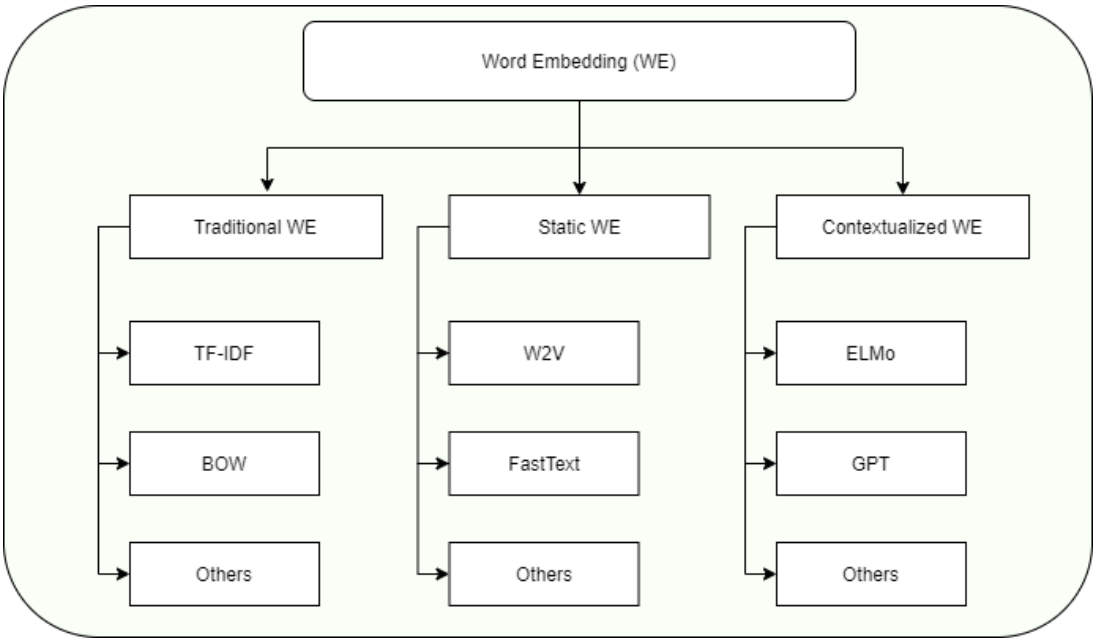

Figure 2.1: Word embedding types.

2.2.1 Traditional Word Embedding

Traditional or frequency-based word embedding is based on a counting technique, i.e. frequency that takes each word and calculates its occurrence in the entire document, discovers the importance of rare words, and estimates the co-occurrence of words. The following are some of the methods that are based on this type.

- **TF-IDF**

  The term TF-IDF stands for "**T**erm **F**requency-**I**nverse **D**ocument **F**requency," and it is defined as a numerical statistic for determining the relative frequency of words in a document in a collection or corpus, which reflects how important the word is in

comparison to the inverse proportion of that word across the entire document corpus [49, 50].
The TF-IDF value rises in proportion to the number of times a word appears in a document but is offset by the word's frequency in the corpus, which helps to regulate the fact that certain words are more common than others by disregarding articles (the, a, an, etc.) and prepositions (in, at, on, etc.) [49].

The TF-IDF is calculated using the equation below [49].

$$TF = \frac{Total\ appearance\ of\ a\ word\ in\ document}{Total\ words\ in\ document} \tag{2.1}$$

$$IDF = log\ \frac{All\ Document\ Number}{Document\ Frequency} \tag{2.2}$$

$$TF - IDF = TF \times IDF \tag{2.3}$$

- **BOW**

Bag-Of-Word (BOW) is a text format that is widely used in machine learning techniques for statistical natural language processing [51].
BOW is dubbed a "bag" of words because it treats the text (sentence/document) as an unordered collection of words by employing a fixed-length sparse vector holding word occurrence counts. The model simply cares about whether or not recognized terms appear in the document, not where they appear [51].
The BOW representation allows a variety of vector-based modeling approaches to be used in NLP applications including document classification and information retrieval [51].

Oscar B et al [52], mentioned that the BOW is only useful for topic-based text classification, not sentiment analysis. By breaking word order and discarding contextual information, the BOW loses contextual information, which is a necessary condition for inaccurate sentiment classification.

2.2.2 Static Word Embedding

Static word embedding is a prediction that method that assigns probabilities to words and maps them to vectors. Training lookup tables that convert words into dense vectors is how static word embeddings learn. The context of this embedding does not change once it has been learned, and the embedding tables do not change between different phrases. Therefore, a word is always mapped to the same vector, and this is what a *"static"* word

returns to [48]. The origins of models based on static word embedding can be traced back to neural language models (NNLMs) because that's how they were made at first [53]. Here are some examples of techniques that are based on this type.

- **W2V**

  Word2vec is a natural language processing approach that is extremely useful in traditional text mining analysis [54], published in 2013 by Mikolov, Sutskever, Chen, Corrado, and Dean [55]. Word2vec aids in the better representation of data by allowing words that are similar to each other to have similar vectors, and words that are not comparable to each other to have separate vectors [54].

  Word2vec can generate a distributed representation of words using one of two model architectures which are continuous bag-of-words (CBOW) or continuous skip-gram [56]. The skip-gram predicts surrounding words given the current word, while the CBOW architecture predicts the current word depending on the context where the order of the words in the context has no bearing on prediction [56].

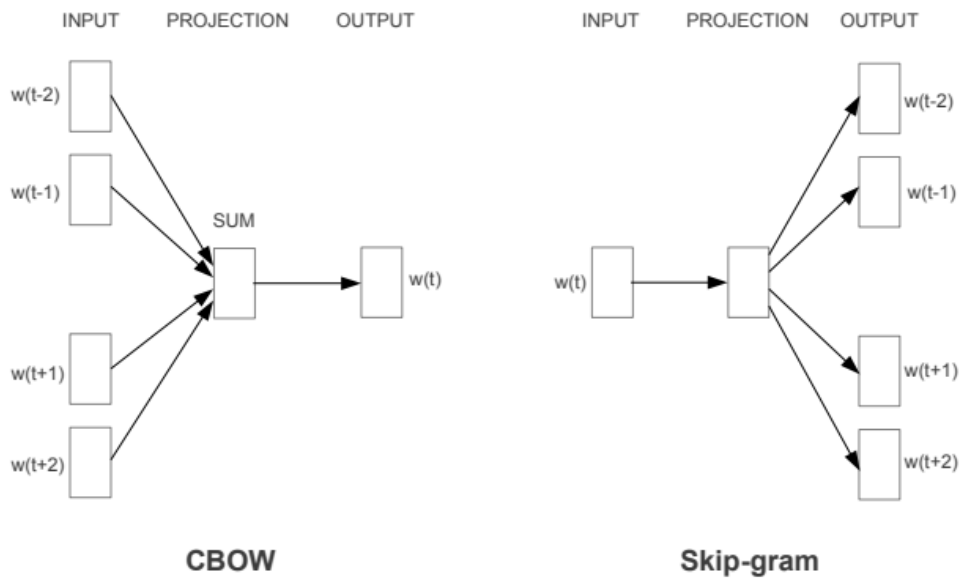

Figure 2.2: CBOW and Skip-gram architectures [55].

- **FastText**

  It is common to underestimate the embedding of uncommon words. As a result, a variety of solutions, including the FastText method, have been created [57].
  FastText, a quick and efficient approach to learning word representations and performing text categorization was just open-sourced by Facebook Research. FastText learns

from both skip-gram and continuous bag-of-words (CBOW), where it based on skip-gram which each word is represented as a bag of character n-grams [58].
Each of the character n-grams has a vector representation, and the word's final representation is the average of these vectors. This paradigm considerably increases performance on syntactic problems, but not so much on semantic issues [57].
The fundamental aim of the fastText embedding is to explore the intrinsic structure of words [58].

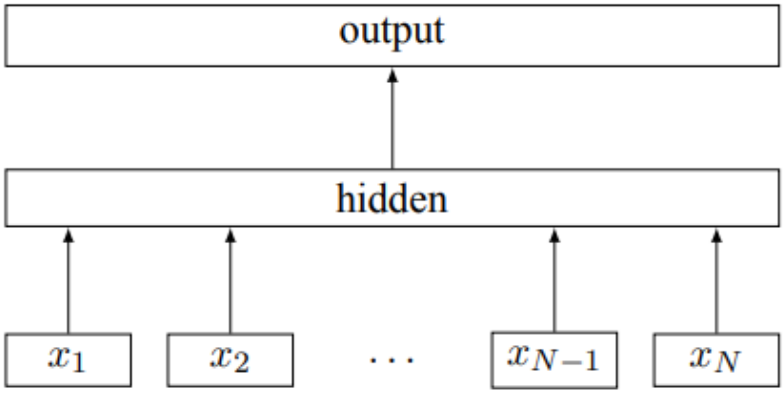

Figure 2.3: Model architecture of Fastext for sentence with n-gram features [59].

2.2.3 Contextualized Word Embedding

Contextual models generate a representation of each word depending on its context in the sentence [60]. As these representations differ according to the meanings of the word and the context used in it, this is what leads to the word can have several representations. Many approaches have become popular, some of them are below.

- **ELMo**

  Peter et al. introduced a new type of deep contextualized word representation and embeddings from language models called ELMo (**E**mbeddings from **L**anguage **Mo**dels) [61], which integrates three neural network layers. A character-based CNN layer, which functions on a character level, is the initial layer. It is context-independent, which means that regardless of the context, each word receives the same embedding. After that, there are two bi-directional LSTMs have been trained on a specific task. In the first LSTM, they try to predict the next word based on the previous words, each represented by CNN's embeddings. The second LSTM is the same as the first, but reading the text backwards. In this layer, they try to predict the previous word using the given next words [61, 62]. ELMo looks at the full sentence before assigning an embedding to each word, rather than utilising a fixed embedding for each word.

The language model in ELMo is bidirectional but only in a "shallow" way [63]. Because even though it binds left-to-right and right-to-left information, it can't bind left and right contexts simultaneously, because the openAI transformer only trains a forward language model.

- **GPT**

The term "**G**enerative **P**re-trained **T**ransformer" (GPT) refers to a set of pre-trained language models produced by OpenAI that has taken the NLP world by storm by presenting extremely strong language models. In Natural Language Generation (NLG) jobs, the most common type of transformer has been GPT [64]. These models are capable of performing a variety of NLP tasks such as question answering, textual entailment, text summarization, and so on.

There are three versions of GPT, where GPT-1 demonstrated that the language model functioned as a successful pre-training target, allowing the model to generalize effectively. The design aided transfer learning and could execute a variety of NLP tasks with minimal fine-tuning.
Concerning GPT-2, it demonstrated that training on a bigger dataset and having more parameters increased the language model's capacity to grasp tasks and outperform the state-of-the-art in many tasks where GPT-2 uses the two-step training strategy of pre-training and fine-tuning [64].
Referring to GPT-3 which is the most extensive language model to present, GPT-3 is trained on a combination of datasets containing 400 billion tokens and has a maximum of 175 billion parameters. In comparison to its predecessor, GPT-3 is capable of few-show learning, in which the model learns from several instances of NLP for NLG tasks termed prompts [64].

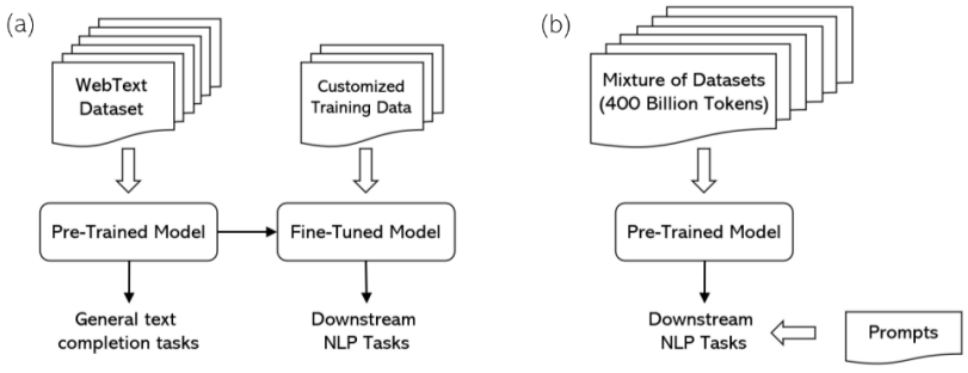

Figure 2.4: Training and re-training process of GPT-2 (a) and GPT-3 (b) [64].

There are many other models and what we are interested in is the giant BERT and

its derivatives. It is a deep contextualised word embedding based on a bidirectional transformer that can be pre-trained and fine-tuned, making it more efficient.

2.3 Word embedding based on deep learning

As previously stated, word embedding deals with the abstract semantic concept of words by capturing lexical semantics in numerical form [65]. Word embedding, which is represented by deep learning, has recently received a lot of attention and is now considered the representational base for a lot of NLP tasks such as text classification, question answering, etc.
We start with a brief overview of deep learning before moving on to the most important models for deep learning-based word embedding.

2.3.1 Artificial Neural Network (ANN)

Artificial **N**eural **N**etwork (**ANN**) is a machine learning algorithm whose idea is inspired by the biological representation of the human brain made up of millions of neurons. Just as biological neurons are the smallest processing unit of biological neural networks that are arranged in a special structure to send and process various electrical and/or chemical signals, artificial neurons also are the basic unit of every artificial neural network, and process information in the same way. We can say that biological neurons and artificial neurons are similar in design and functionalities. ANN is made up of many interconnected neurons which contain activation functions associated with them [66]. The activation functions for a node define the relation between the output of that node with a set of inputs [67].
A simple artificial neural network is made up of several layers of neurons, including an input layer, a hidden layer, and an output layer.

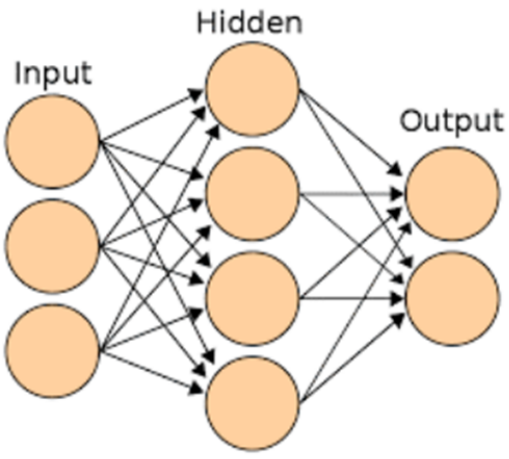

Figure 2.5: Artificial neural networks architecture.

In short, the network receives data via the input layer, which then communicates with other layers. This data is processed by multiplying it by weights, then summing its results, applying the activation function to it, and producing the output signal [66]. In the end, an artificial neuron passes the processed information through the output layer.

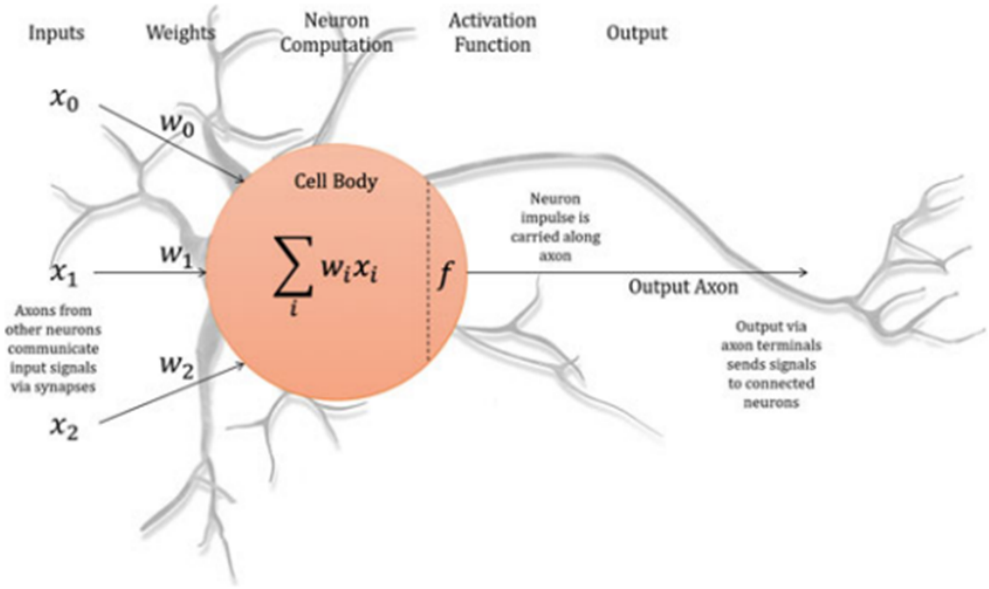

Figure 2.6: Diagram of an artificial neuron [68].

2.3.2 Deep Networks

Deep **L**earning (**DL**) is one of the most talked-about concepts in artificial intelligence nowadays. In fact, there is a lack of a unified definition of it in literature. Some said that "deep learning is a method that aims at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower-level features" [69]. According to [70], "Deep learning is a process not only to learn the relationships among two or more variables but also the knowledge that governs the relation as well as the knowledge that makes sense of the relation.". They also added that their proposed definition is not limited to a hierarchy of abstractions or representations or concepts but a network as well. Deep learning refers to the deep neural network, which is a specific configuration where neurons are arranged in multiple consecutive layers [71]. In general, deep networks are still neural networks but commonly with more layers. Its hallmark is that through its evolution, it can be applied to problems that were not previously possible with traditional methods and smaller neural networks [68].

Deep neural networks architectures

The precise design of network architecture is one of the factors that determine the success of neural networks. Although deep structures have a higher training time than ANN, deep learning structures perform better than simple ANNs. Some of the popular and widely used deep learning architectures are discussed below.

- **Convolutional neural networks (CNN):** CNNs are a specific type of artificial neural network that is widely used in various applications including: face detection, image processing, voice recognition and even NLP [72]. Its most important benefit is that it determines the relevant features without any human supervision [73]. Its architecture consists of a number of main layers which are: the convolution layer, pooling layer, activation function (or non-linearity) and fully connected layer. The convolutional and fully-connected layers have parameters but pooling and non-linearity layers have no parameters [74]. For a better understanding, you can check those articles [73, 74].

- **Recurrent neural networks (RNN):** RNNs have attracted a lot of interest. It is a class of deep artificial neural networks that deals with sequences, except that each neuron can use its memory to remember information about the previous step. It is a network that uses a hidden state h to handle a variable-length input sequence $x = (X_1,...,X_t)$ and optionally a variable-length output sequence $y = (Y_1,...,Y_t)$ [75]. Tasks related to time series data, sequential data and most NLP tasks are based on this network. As for depth, it is the same as the length of the input data sequence [76].

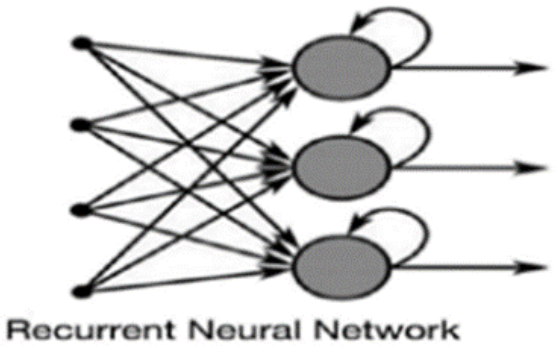

Figure 2.7: Recurrent Neural Network [77].

An RNN does the same operations on every single element of the input sequence since it processes the sequence information. Understanding how it works is very difficult. According to the above figure, its output, at each time step, depends on previous inputs and previous calculations. This gives it the advantage of memory for past events, and it is in the hidden layers of this network [78]. But RNN finds it difficult to track long-term dependencies i.e. when the gap between the relevant input data is large [79].

- **Bidirectional recurrent neural networks (BRNN):** Bidirectional recurrent neural networks (BRNN) connect two hidden layers in opposite directions to a single output. The model is able to exploit information from both the past and the future [80]. BRNNs are especially useful when the context of the input is needed. For example, in handwriting recognition, the performance can be enhanced by knowledge of the letters located before and after the current letter.

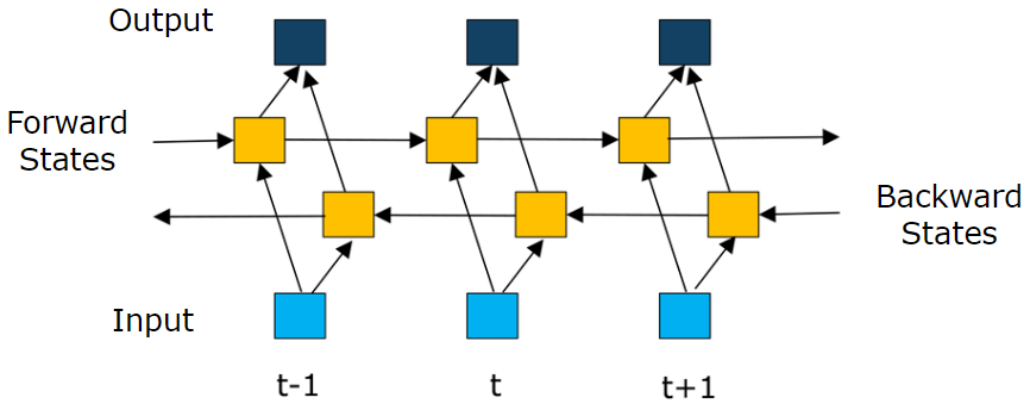

Figure 2.8: Bidirectional recurrent neural networks.

- **Long short-term memory (LSTM):** The above RNNs are not able to connect important information when the gap between the relevant input data is large, for example, in the case of long sentences and paragraphs that contain too many words between the nouns and the verb. This is where the idea of LSTM networks came in, a type of RNN capable of dealing with long-term dependencies. In addition to achieving almost all the results of RNN, it also has a strong learning ability dependencies i.e. when the gap between the relevant input data is large [79], and has been adopted in many tasks such as speech recognition [81] and machine translation [82]. Hochreiter and Schmidhuber [83] first introduced the LSTM cell in 1997 by inserting a "gate" into the cell (input gate, output gate). Then it was modified to LSTM with a forget gate, and LSTM with a peephole connection. Usually, the term LSTM cell refers to an LSTM with a forget gate [79].

  The learning capability of the LSTM cell is strong, but the additional parameters augment the computational burden, so the gated recurrent unit (GRU) has been proposed.

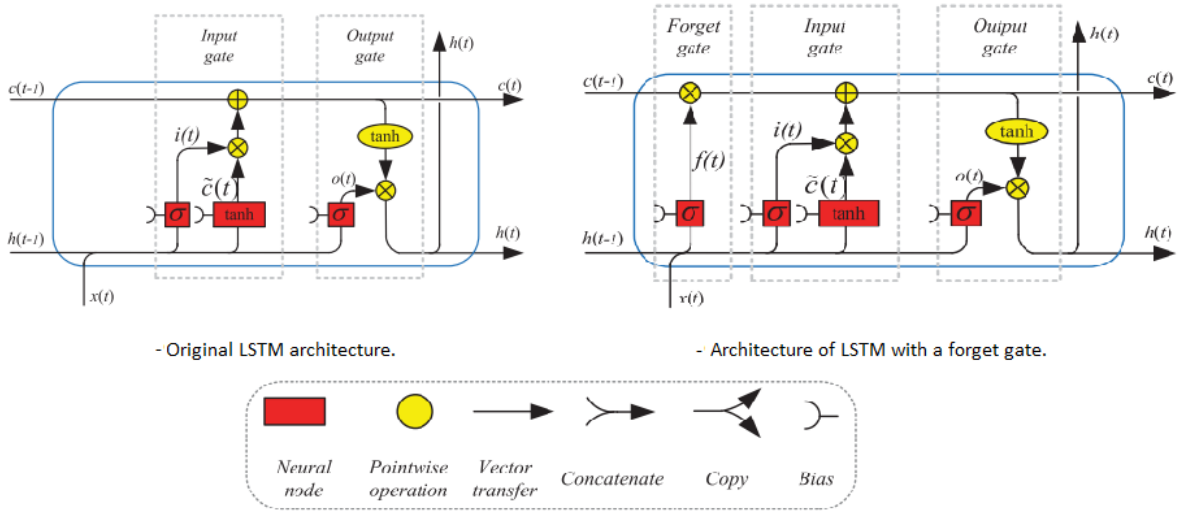

Figure 2.9: LSTM architecture [79].

- **Gated recurrent units (GRUs):** GRU is a new RNN architecture that was proposed in 2014 by Kyunghyun Cho et al. [84]. The GRU reduces the number of parameters by integrating the forget gate and the input gate of the LSTM cell with the forget gate as the update gate. Thus, it has only two gates, a reset gate and an update gate [79].

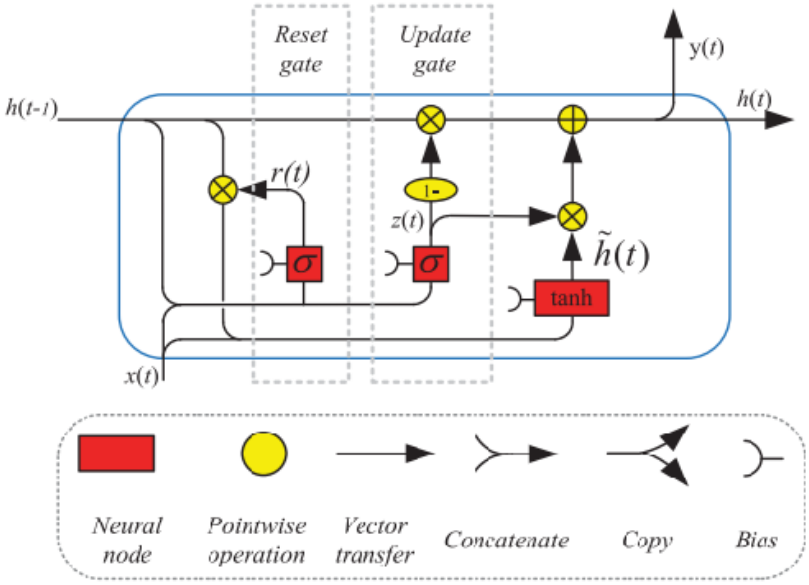

Figure 2.10: GRU cell architecture and connections. [79].

2.3.3 Sequence-to-sequence models

Despite the flexibility and capability of DNNs, they can only be used to solve problems when the inputs and outputs can be coherently encoded using fixed-dimension vectors.

They are not suitable for situations where the sequence length is unknown in advance. For instance, consider text summarization, which takes a large string of words as input and produces a concise summary that is likewise a sequence.
The idea was to use an LSTM network to read the input sequence one timestep at a time, resulting in a large, fixed-dimensional vector representation, and then extract the output sequence from that vector using another LSTM. This is called the sequence-to-sequence model. [85].

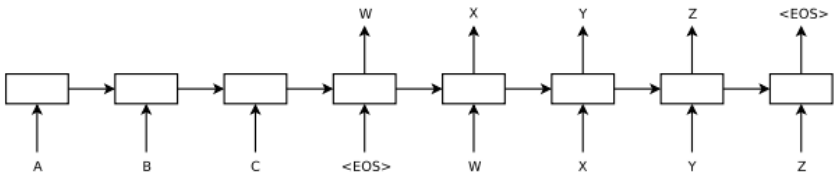

Figure 2.11: Sequence-to-sequence architecture.

Sequence-to-Sequence models combine two common deep learning architectures, recurrent neural networks and the encoder-decoder model which are commonly used to solve complex language problems. Both encoder and decoder are LSTM models (or sometimes GRU models).

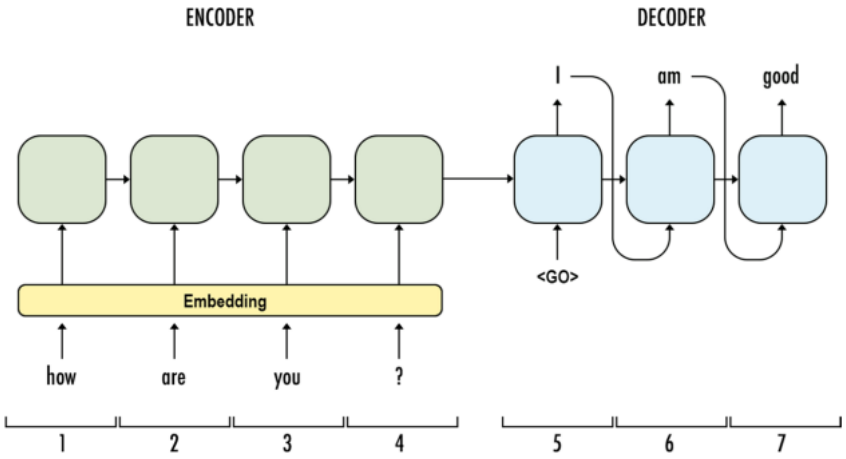

Figure 2.12: Encoder-decoder model.

In case of text summarization, during training, the source text and reference summary data are tokenized and given to the encoder and decoder networks. The encoder network reads the source text and converts it to a context vector, which is then passed on to the decoder network, whose initial states are initialised by the encoder's final states (for more details refer to [86]), to aid in the prediction of the summary sequence.

2.3.4 Attention mechanism

Philosopher William James said about attention in his book [87]: "it implies withdrawal from some things in order to deal effectively with others...". On the same principle, the attention mechanism works. It was derived first from human intuition. Then by 2014, the DL community noted that this mechanism represented a fundamental notion for the advancement of deep neural networks [88]. It was first adopted in the field of natural language processing (NLP) for machine translation tasks by D.Bahdanau [86]. Currently, it is widely applied, and the latest technologies in this field use attention mechanisms, including sentiment classification, text summarization, etc.

The attentional mechanism is a part of the neural architecture that allows highlighting relevant aspects of the input i.e. it helps us to evaluate the importance of these elements (by calculating the weight distribution over the input sequence, assigning higher values to more relevant elements [89]) and compacts them into a representation that condenses the features of the most relevant parts, called a context vector [89]. In case of NLP, the input is usually a series of text items.

Various structures of attention have been proposed in recent years, such as multi-dimensional attention and memory-based attention, but we will only address self-attention, or as it is called intra-attention, as it has become a well-established building block of neural methods in NLP. It has been used for many tasks such as abstractive summarization. Simply self-attention is an attention mechanism that measures the relationship between elements of a single sequence in order to calculate the representation of the same sequence [90]. The inputs interact with each other, "self", and decide what to give more attention [88], to capture contextual information deep within the sentence [91].

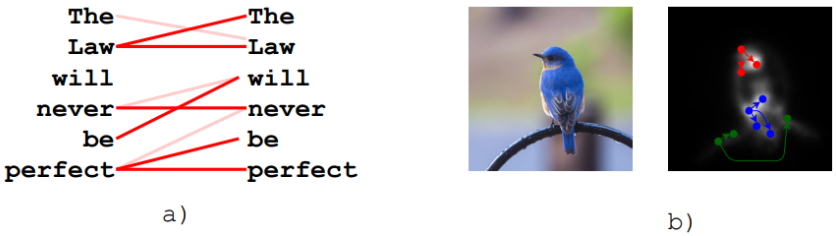

Figure 2.13: Self-Attention examples. a) Self-attention in sentences b) Self-attention in images. The first image shows five representative query locations with color-coded dots with the corresponding color-coded arrows summarizing the most-attended regions [88].

2.3.5 Transformers

The transformer has gone beyond neural models such as convolutional and recurrent neural networks in performing natural language understanding and natural language generation tasks, becoming the dominant architecture for NLP [92].
The Transformer [90] is a model architecture in which every output element is connected

to every input element, completely dependent on the self-attention mechanisms. Transformer blocks are composed of two components: 6 layers of encoders and 6 layers of decoders. Each layer of both has two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. But the decoder has a third sub-layer which helps it to focus on the output of the encoder stack. Residual connections [93] connect each of the sub-layers in both the encoder and decoder followed by layer normalization [94].

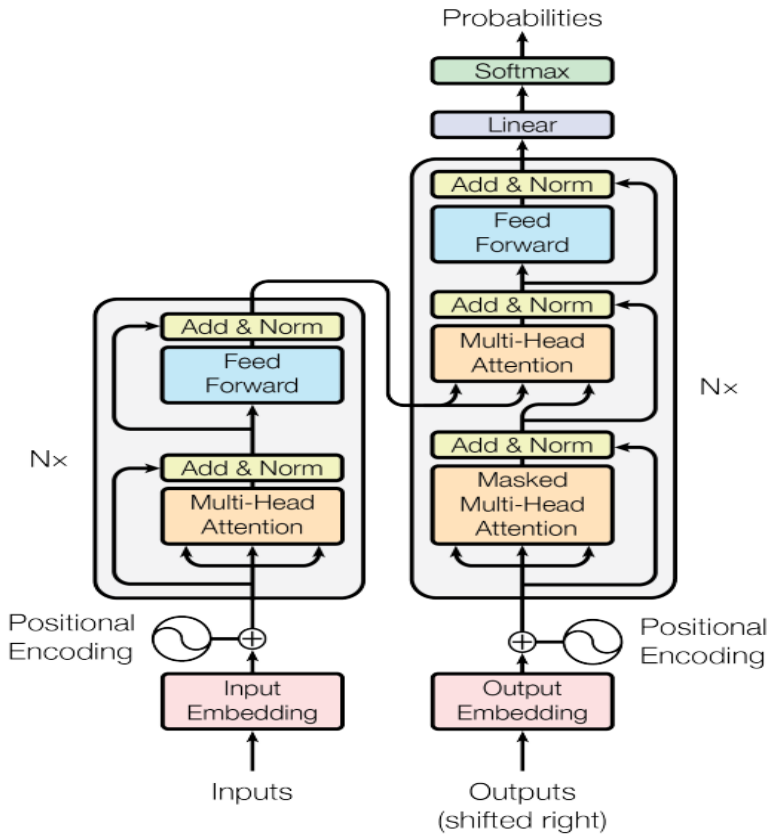

Figure 2.14: The Transformer - model architecture [90].

2.3.6 BERT

A new era in natural language processing began with the birth of BERT [95], or **B**idirectional **E**ncoder **R**epresentations from **T**ransformers, a model that excelled in how well models handled language-based tasks. A significant change in the way Google's search engine and its translator work has been observed since adopting this model in late 2019. BERT, open-sourced and trained by Google, is a method of pre-training language representations. It is the first deeply bidirectional, unsupervised system, trained only using a large set of plain text (BooksCorpus (800M words) and English Wikipedia (2,500M words)), then used for NLP tasks (such as question-answering) where it got state-of-the-art results outperforming previous unidirectional methods.

It seems that Google's goal was to understand the user's intent rather than his query

words, so he made the model process texts in both left-to-right and right-to-left directions simultaneously to predict the intended meaning of a word depending on the previous and next context in the sentence it is in. As an example of this, we take the word "bank" from the sentence "I made a bank deposit". BERT represents "bank" using both "I made a" and "deposit". In this way, it can distinguish between representing a "bank deposit" and a "river bank". Google calls BERT "deeply bidirectional" because contextual representations of words start from the bottom of a deep neural network [63]. Figure 2.15 illustrates a comparison between the visualization of the neural network architecture of BERT with previous contextual pre-training methods. Based on the arrows that show the direction of information flow from the bottom to the up, it is found that BERT is deeply bidirectional, in contrast to OpenAI Generative Pre-Training (GPT) which is unidirectional [96], while Embeddings from Language Models (ElMo) is shallowly bidirectional [61].

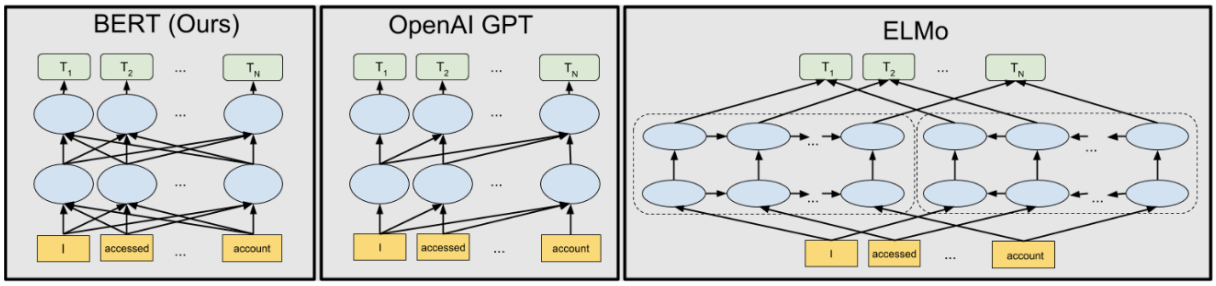

Figure 2.15: Bi-directional BERT compared to others [97].

However, training bidirectional models results in the problem that the word to be predicted indirectly "see itself" in a multi-layer model and the model can therefore trivially predict the word. To solve this problem, the **M**asked **L**anguage **M**odel (MLM) task was used. 15% of the words in the input are randomly masked, and then the whole sequence is fed through a deep bidirectional Transformer encoder with the objective that the masked words are only predicted based on context. For example:

Input: the man went to the [MASK1]. he bought a [MASK2] of milk.

Labels: [MASK1] = store; [MASK2] = gallon.

In addition to being trained on the MLM, BERT was pre-trained using an unsupervised task named **N**ext **S**entence **P**rediction (NSP) in order for the model to be able to understand the relationship between two sentences. This happens by generating sentences from any monolingual text corpus, then selecting two sentences A and B: 50% of the time B is the next actual sentence following A labelled as *IsNextSentence*, and 50% of the time it is a random corpus sentence labelled as *NotNextSentence*. For example:

Sentence A: the man went to the store.

Sentence B: he bought a gallon of milk.

Label: IsNextSentence

Sentence A: the man went to the store.

Sentence B: penguins are flightless.

Label: NotNextSentence

To summarize the above, BERT has been pre-trained on massive amounts of text on two unsupervised tasks which are: MLM and NSP, using a large model consisting of a 12 layers transformers for BERTBASE (and 24 layers for BERTLARGE). Creating state-of-the-art models with BERT requires fine-tuning it with only one additional output layer after it has been initialized with pre-trained parameters and then fine-tuning those parameters using labelled data from the final tasks [95].

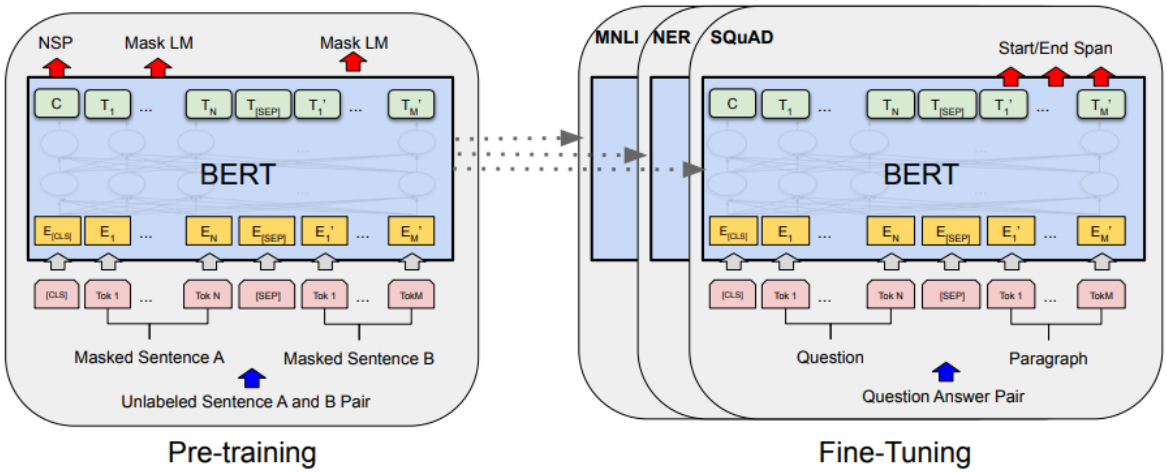

Figure 2.16: Overall pre-training and fine-tuning procedures for BERT [95].

Figure 2.16 from the BERT paper shows that the same structures are used in both pre-training and fine-tuning except for the output layers. Each downstream task ultimately has its own fine-tuned models, even though they are initialised using the same pre-trained parameters.

As for the BERT inputs and outputs, they used the WordPiece embeddings [82] with a 30,000 token vocabulary. Pairs of sentences are grouped in a single sequence. The sequence begins with a special classification token ([CLS]) and each sentence is separated by a special token ([SEP]) e.g. separating questions/answers as shown in Figure 2.16. Each token has a learned embedding demonstrating which sentence A or B belongs to, as shown in the figure. This method allows a single sentence or a pair of sentences to be represented clearly and unambiguously in a single sequence with the intent that BERT can handle a variety of downstream tasks. For each token, its input representation is the sum of the corresponding token embeddings, the segmentation embeddings and the position embeddings, as displayed in Figure 2.17.

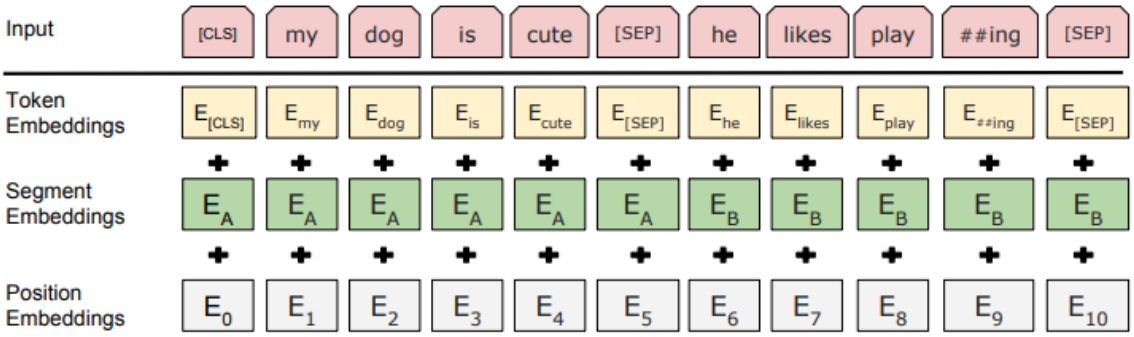

Figure 2.17: BERT input representation [95].

Limitations of BERT

- **The sequence's length:** BERT's inability to handle long text sequences is one of its shortcomings. By default, BERT accepts up to 512 tokens.

- **Tokenization:** At every sub-word unit, BERT tokenizes the longest in-vocab sub-string from the left. While this works effectively for words with suffixed roots (or stems), prefixed terms suffer from poor tokenization. According to Anmol Nayak et al. [98], even though the vocabulary included the prefixes *de* and *un*, as well as the terms *constructed*, *activated*, and *equal*, the tokenizer chose the sub-strings *deco*, *dea*, and *une* for *deconstructed*, *deactivated*, and *unequal*.

BERT models

As previously stated, BERT was trained on BooksCorpus and the English Wikipedia, and two models were provided (BERTbase with 12 layers transformers and BERTlarge with 24 layers), following which numerous well-known and frequently used models for the English language, as well as other languages, including Arabic, appeared. Some of them are listed below.

- **RoBERTa**

  For **R**obustly **o**ptimized **BERT** **a**pproach [99], improves on BERT's pretraining approach by training the model for longer periods of time with larger batches of data, removing the next sentence prediction target, training on longer sequences, and dynamically altering the masking pattern applied to the training data.

- **DistilBERT**

  A general-purpose pre-trained version of BERT, was introduced by Victor Sanh et al. [100]. It contains 40% fewer parameters and runs 60% quicker than BERTbase while maintaining 97% of its language understanding capabilities. Knowledge distillation was

used to train DistilBERT, which is a technique for compressing a huge model termed the teacher into a smaller model named the student. They got a smaller transformer model by distilling Bert, which retains many similarities to the original BERT model while being lighter, smaller, and faster to run.

- **AraBERT**

  AraBERT, built by Wissam Antoun, Fady Baly, and Hazem Hajj [101], is an Arabic pre-trained language model based on the BERTbase architecture. It comes in 6 variants, including, AraBERTv0.2 base and large, AraBERTv2 base and large, AraBERTv0.1 base and large, and other new models for Arabic dialects and tweets called AraBERTv0.2-Twitter base and large too. All those models are under this repository [102].

2.3.7 BART

BART is a model that combines **B**idirectional and **A**uto-**R**egressive **T**ransformers and pre-trains it. It is a denoising autoencoder based on a sequence-to-sequence model that can be used to perform a vast range of tasks. BART learns to recreate the original text after corrupting it with an arbitrary noising function. The noising function for generation tasks was text infilling, which used single mask tokens to mask randomly sampled text spans. It employs a basic Transformer-based neural machine translation architecture. It is almost a combination of generalised BERT (due to the bidirectional encoder) and GPT (due to the left-to-right decoder).

In Figure 2.18, the document is bidirectionally encoded, with random tokens replaced by masks. BERT can't easily be utilised for generation since missing tokens are predicted independently.

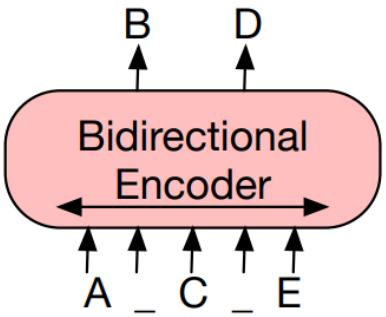

Figure 2.18: BERT [103].

As for GPT (Figure 2.19), tokens are auto-regressively anticipated, which means GPT can be employed for the generation. Words, on the other hand, can only condition in a leftward context, so bidirectional interactions are impossible to learn.

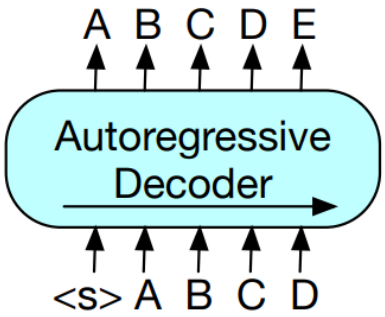

Figure 2.19: GPT [103].

For BART in Figure 2.20, encoder inputs do not have to match decoder outputs, allowing for arbitrary noise modifications. A document has been corrupted by mask symbols being used to replace text spans. The probability of the original document (right) is computed using an autoregressive decoder after the corrupted document (left) is encoded with a bidirectional model. An uncorrupted document is used as an input to the encoder and decoder for fine-tuning, and representations from the decoder's final hidden state are employed.

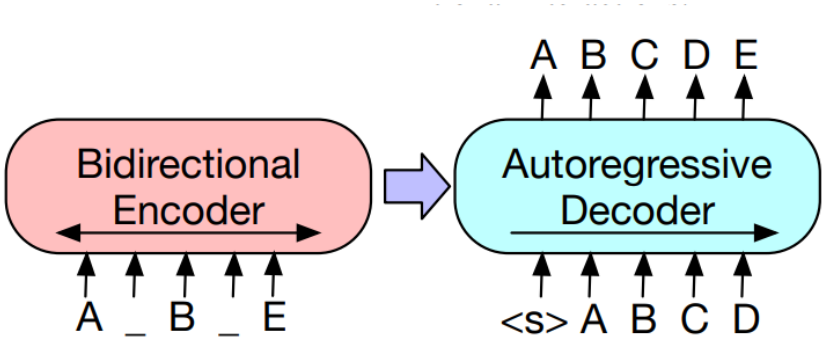

Figure 2.20: BART [103].

The base model has 6 layers in the encoder and decoder and the large model has 2 layers in each. The architecture is similar to that used in BERT; however, BART has around 10% more parameters than the BERT model of the same size.

BART was trained by corrupting the text with a random obfuscation function. It allows applying any type of document corruption. The modifications that were used in the experiments are summarized below (Figure 2.21), which are token masking, token deletion, text infilling, sentence permutation, and document rotation.

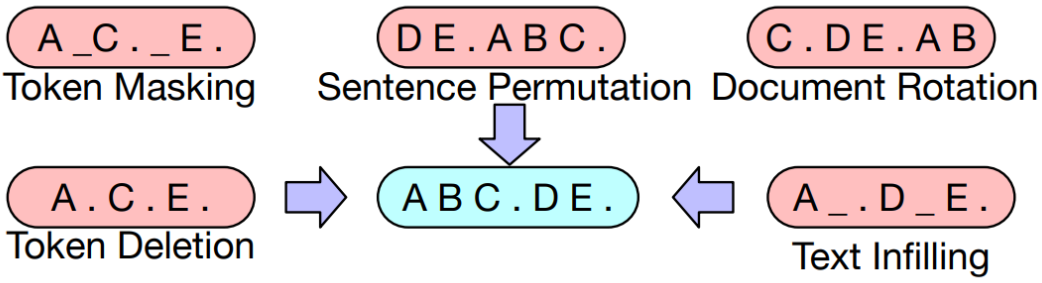

Figure 2.21: Transformations for noising the input that were experimented with. [103].

BART's representations can be utilised for a variety of downstream applications, such as abstractive question answering and summarization. BART can be easily fine-tuned for sequence generation applications because it features an autoregressive decoder. The information is copied from the input but altered in both of these tasks, which is strongly related to the pre-training goal of denoising. The input sequence is encoded by the encoder, and the outputs are generated by the decoder.

2.3.8 PEGASUS

Pre-training with **E**xtracted **G**ap-sentences for **A**bstractive **SU**mmarization **S**equence-to-sequence models, or **PEGASUS** [104], trains a transformer encoder-decoder model to increase fine-tuning performance on abstractive summarization under the idea that the closer the pre-training self-supervised objective is to the final task, the more useful the fine-tuning performance.

A self-supervised objective **G**ap **S**entence **G**eneration (**GSG**) in PEGASUS pre-training was used. Similar to extractive summarization, some full sentences are removed or masked from the input documents and the model is charged with generating them as one output sequence from the remaining phrases.

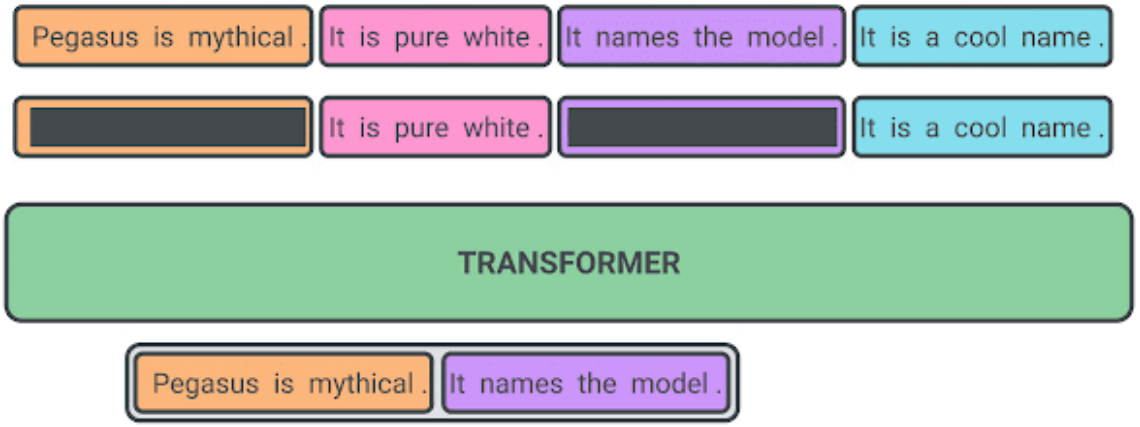

Figure 2.22: Example of pre-taining PEGASUS on GSG [105].

Various scenarios for selecting the gap sentence have been studied and the ideal strategy is to select "important" sentences to mask it (the ones most similar to the rest of the document according to the ROUGE scale), which increases the resemblance of self-supervised examples' output to a summary.

After the model had been pre-trained on a huge number of web-crawled texts, it was put to fine-tuning on different abstractive summarization datasets including scientific papers, patents, short stories, emails, and legal documents. According to ROUGE and human review, Pegasus performs state-of-the-art summarization performance in 12 different summarization datasets.

2.4 Conclusion

In this chapter, we have covered some of the most crucial fundamentals of our current work, including deep learning, and word embeddings. BERT, BART and PEGASUS were chosen as the chapter's main focus since they are the core issue of this study and have a relation to deep word embedding.

In the following chapter, we will go over the system's design in addition to explaining its main components, as well as how to prepare data for the model used.

Chapter 3

System design

3.1 Introduction

In fact, the complexity of the Arabic language in terms of morphology and other aspects makes it difficult to build models that support it. From the recent past to the present, deep contextualized word embedding, especially BERT and its derivatives, have been demonstrated to function effectively in the NLP domain. We tried to take advantage of this feature to summarize the Arabic document.

We used two different methods in this work. One is based on BART, which covers fine-tuning one of the Arabic models to accomplish the summarization task. As for the second, it is the use of one of the most successful ready-made fine-tuned models for summarizing which is PEGASUS.

In the current chapter, we will provide an overview of the design architecture used to construct the abstractive Arabic text summarizer.

3.2 Arabic text preprocessing

The process of preparing and cleaning raw data from impurities such as unwanted characters and symbols in order to make it suitable for building and training machine learning models is known as data preprocessing. To prepare the raw text data, segmentation and normalization processes are applied. Text segmentation is where the text is divided into sentences or sentences are divided into words as needed. Normalization refers to reducing the orthographic variation. The several aspects of cleaning raw data that can be done include the following:

- **Removing diacritics and kashidas from the Arabic text:** The diacritic is computationally expensive, but removing it increases the ambiguity of the text, especially for words with the same letters and different meanings such as (جبن) means cheese and (جبن) means cowardice. However, this ambiguity does not constitute an obstacle in the case of using BERT, because it recognizes the intended meaning from the context of the sentence. As for deleting the kashidas, it is with the aim of returning the word to its natural form.

- **Elimination of common misspellings:** These errors are made when replacing, for example, (ى) with (ي) or (ة) with (ه). To solve this problem, some people map all (ي) to (ى), (ة) to (ه) and change the different forms of the Arabic hamza (إ,أ,آ,ا) to one (ا).

- **Stemming:** recover the main units of meaning. From a morphological point of view, some people prefer to return words to their original form, as they were before their conjugation and remove the prefixes and suffixes from them. The word becomes a stem (or root)

- **Other aspects of data preprocessing:** can be done as needed, such as removing HTML tags, punctuation, Latin letters, and URLs.

- **Tokenization:** In the case of deep contextualized word representation, the raw text is incomprehensible to the model. It requires another process that converts the data into a format that it can understand. This process is called tokenization. It is the

transformation of text data into numerical values by breaking text into tokens and converting tokens to their matching IDs in the pre-trained vocabulary.

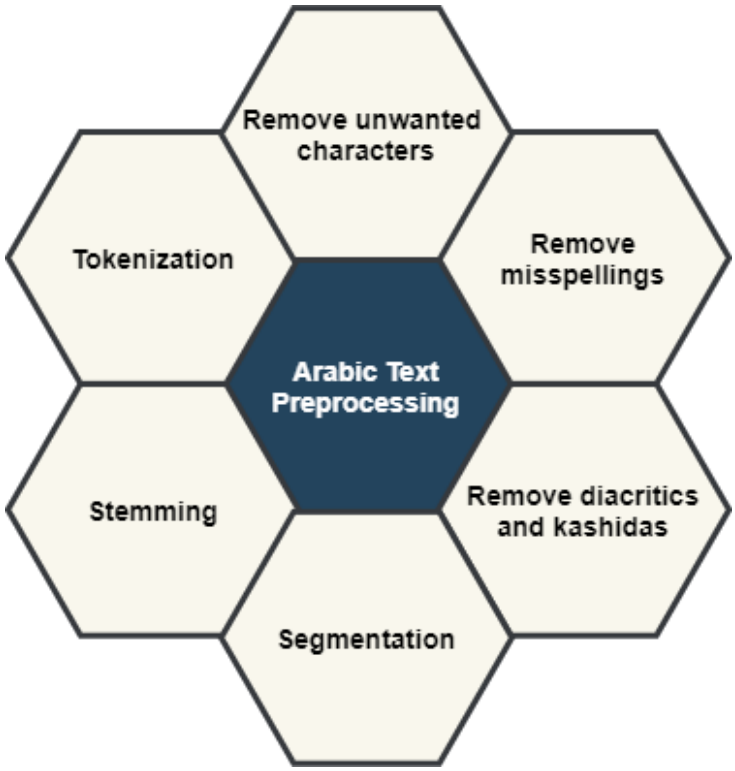

Figure 3.1: Arabic text preprocessing aspects.

3.3 Text summarizer based on BART

The schematic below presents the general architecture of the text summarizer, showing the set of phases that will follow, the system inputs and the outputs, as we represent in Figure 3.2.

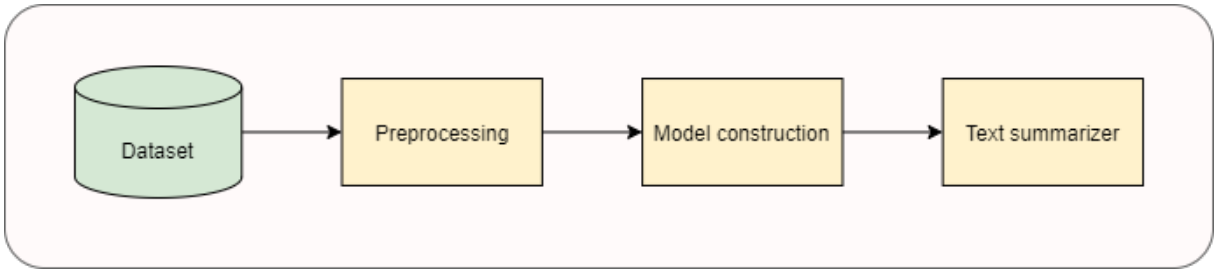

Figure 3.2: Diagram of the text summarizer based on BART.

3.3.1 Model construction

The pre-trained model BART feeds on the data intended for training, occurs so-called fine-tuning, and is ready for summarization at this point. Then it sends test data into the summarizer to summarise its texts. The process is depicted in Figure 3.3.

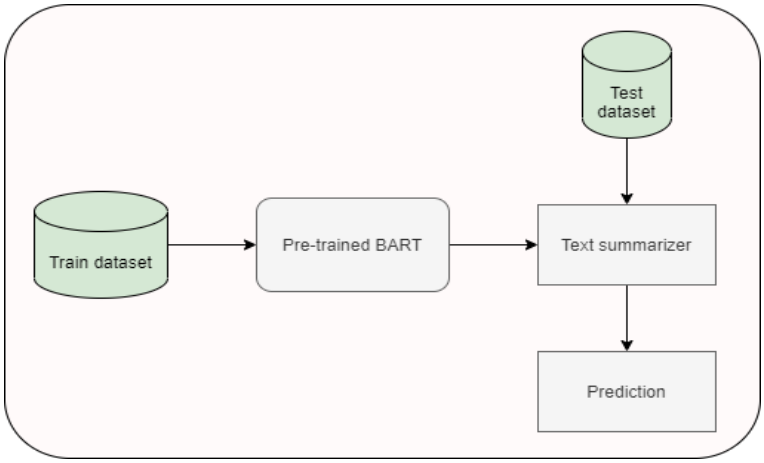

Figure 3.3: Model construction process

Fine-tuning

Creating models for NLP tasks based on pre-trained ones requires fine-tuning. It's a very effective training method that involves taking a model that has already been trained for a particular task and tuning or tweaking it to accomplish a second task.
Fine-tuning is done by training the pre-trained model on a specific task labelled dataset using the same structures and parameters that were used in the pre-training. Even though they are all initialised using the same pre-trained parameters, each downstream task eventually has its own fine-tuned models.

In some cases, it may require the removal of only one or more final layers and the addition of one or more layers on the top layers, each depending on how similar the task to be obtained and the task on which the model was trained.

Pre-trained models can be fine-tuned for a variety of advantages, the most important of which are:

- **Faster development, less computation:** The training time is significantly reduced because the fine-tuned model takes less time to train. This is due to the use of pre-trained weights for the model's first layers. Thus our network is fully trained and only needs to train the final layers, as a result of which the computation cost is reduced.

- **Less data, less effort:** Pre-trained models are trained on massive, large-scale

datasets. Since the model's performance improves with more training data, it is sure that getting an effective model requires an exorbitantly large data set, Which requires a lot of time and effort to collect data. The pre-trained models cames with a good solution, which is to be used without having to train one from scratch, just fine-tuning them using pre-trained weights on a much smaller dataset.

Pre-trained BART

Transformers gives access to thousands of pre-trained models for a variety of tasks. AraBART is the first Arabic model based on BART in which both the encoder and decoder are pre-trained end-to-end. It is based on the BART-Base architecture, which comprises 6 encoder and 6 decoder layers, as well as 768 hidden dimensions. There are 139M parameters in AraBART in total. This model outperforms strong baselines, such as pre-trained Arabic BERT-based models and multilingual mBART and mT5 models, on multiple abstractive summarization datasets. We feed this model with preprocessed training and validation data to train it on the summarization task.

3.3.2 Prediction phase

There is a test phase that follows the training. After the model has been trained on the dataset multiple times, it is ready to summarize texts. We bring an Arabic text to which we apply preprocessing and give it to the model as input. It generates a prediction in the form of tokens, i.e. numbers. These given numbers are IDs. We decode them to retrieve the summary generated by the fine-tuned model by recovering the words matching these IDs from the pre-trained vocabulary, which is the opposite of the tokenization process that took place in the data preprocessing stage.

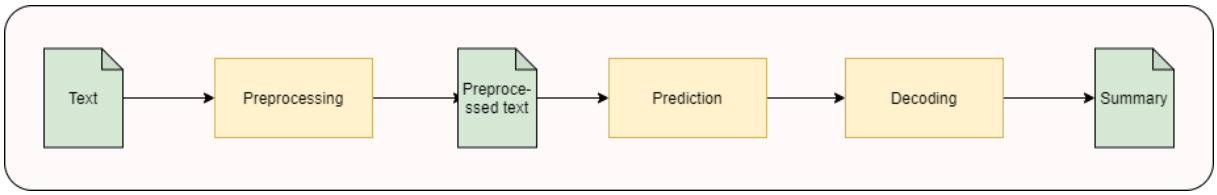

Figure 3.4: Process of the prediction phase.

3.4 Text summarizer based on PEGASUS

The second text summarizer that was suggested is based on google PEGASUS, and its architecture is shown in Figure 3.5.
This figure embodies the steps that the system follows to be able to perform the summarization of the Arabic document. Starting with the data, it is preprocessed and sent to

the summarization process. Some sequential operations are applied to it to end it in the form of a summary.

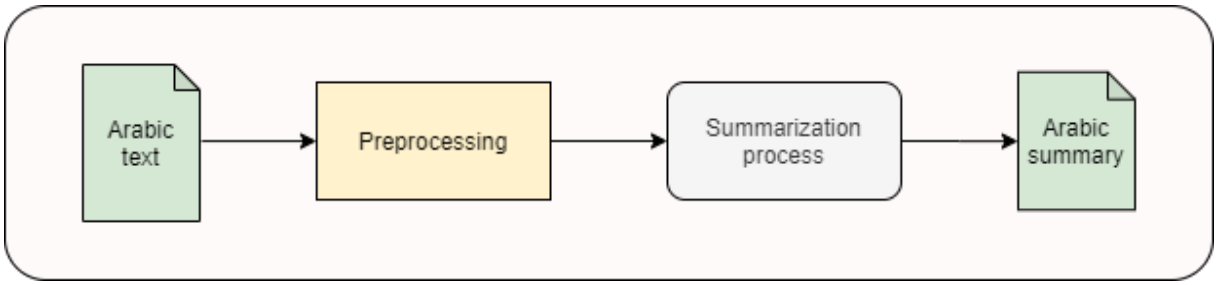

Figure 3.5: Diagram of text summarizer based on PEGASUS.

Summarization process

To the best of our knowledge, PEGASUS does not support the Arabic language, so we took a way to summarize the Arabic document using it. We added two translation layers before and after the summarizer. Before accessing PEGASUS, the data must first pass through the first layer, where it is translated from Arabic to English, so that it becomes accepted by the summarizer. PEGASUS does its task by generating summaries as output, which are in turn inputs to the next translation layer. The summaries are translated and returned to the Arabic language. Figure 3.6 shows an overview of these steps.

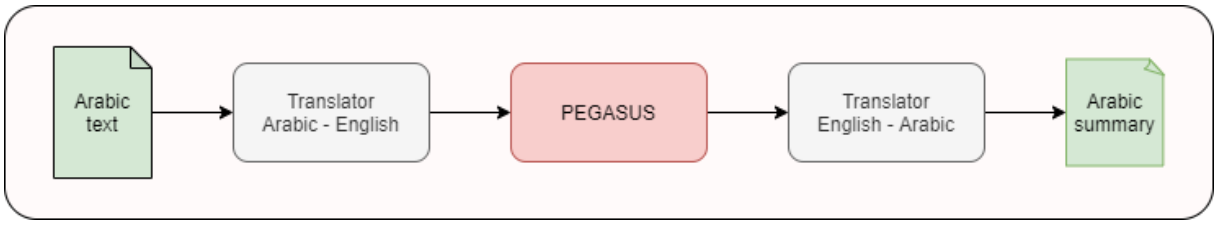

Figure 3.6: Diagram of text summarizer based on PEGASUS.

3.5 Conclusion

The purpose of this chapter is to provide a wide and comprehensive description of the abstractive summarizer structure, including the preprocessing of datasets and preparation steps.

In the next chapter, we'll go through the various tools, frameworks, and libraries that we have utilized, as well as describe the implementation of our two systems, show their results and compare them.

Chapter 4

Implementation and results

4.1 Introduction

After describing our systems design in the previous chapter and offering an overview of our models, we will move on to the core of our work. This chapter will cover the working environment, programming language, and tools we utilized to develop the systems. After that, we will report the results of the two summarizers. In the end, a comparison will be made between the summaries of both systems using the ROUGE metric.

4.2 Implementation frameworks and tools

A variety of technologies can be used to accomplish the deep natural language processes. We can find these open-source programs on the internet. This section details the many development frameworks and tools that were employed in the creation of this theme.

4.2.1 Python

Python is the most widely used and well-known programming language in data science. It is an interpreted, object-oriented and high-level programming language that is supposed to be simple to read and use, because of its simplest and easy-to-learn syntax. It's free to use, even for commercial purposes, because it's open-source. Python facilitates programme flexibility and code reuse by supporting modules and packages [106].
This programming language is used in many application domains such as Data Science (like our case), Web and Internet Development, Software Development, Business Applications, etc.
The python version that was utilised in this project is 3.7.12 as shown in Figure 4.1

```
import sys
print("Python version:"+sys.version)
```

```
Python version:3.7.12 | packaged by conda-forge | (default, Oct 26 2021, 06:08:53)
[GCC 9.4.0]
```

Figure 4.1: Version of python used in this work.

4.2.2 Kaggle

It is a competition platform that became for Google as of March 2017, that enables other data scientists and developers to participate in ongoing machine learning competitions, write and share code, and host data sets to solve problems in data science, machine learning, and predictive analytics. Kaggle Kernels supply a free Cloud GPU with 30 hours weekly.
This platform provides contributors with intriguing and challenging projects in which they can learn and apply their skills. In addition, there will be enlightening talks with industry leaders and knowledgeable specialists. Kaggle invites its audience to become a part of the world's largest data science community [107].

4.2.3 PyTorch

PyTorch is a Python-based machine learning package that focuses on natural language processing. It is a tensor library intended for usage with GPUs and CPUs in Deep Learning applications. The open-source program was created by Facebook AI artificial intelligence teams in 2016. Tensor computing and functional deep neural networks are two of PyTorch's most notable capabilities [108].

Figure 4.2: PyTorch logo.

4.2.4 Transformers

Transformers offers APIs that make it simple to download and train state-of-the-art pre-trained models. Using pre-trained models can help you reduce compute costs and time by eliminating the need to train a model from scratch. The models can be used in a variety of modalities, including text (text classification, question answering, summarization, etc.) in over 100 languages, images (image classification, object detection, etc.), audio (speech recognition, etc.), and multimodal (information extraction from scanned documents, video

classification, and so on). PyTorch, TensorFlow, and JAX are three of the most prominent deep learning libraries, and this library provides smooth integration between them [109].

Figure 4.3: Transformers logo.

Transformers version used here is 4.18.0 as illustrated in Figure 4.4.

```
import transformers
print("Transformers version:"+transformers.__version__)
```

```
Transformers version:4.18.0
```

Figure 4.4: Version of transformers used in this work.

4.2.5 Deep translator

A versatile, unrestricted tool for translating between languages in a straightforward manner employing many translators. It has many features, including, it's the only Python tool with multi-language support and many translators, it's simple to use and extend with automatic language detection [110].

Figure 4.5: Deep translator logo.

4.2.6 Tqdm

Tqdm is a Python package that wraps around any iterable and outputs a clever toolbar. A tqdm progress bar not only displays the amount of time that has passed but also the estimated time that the iterable will take to complete.

4.2.7 Pipelines

The pipeline is a notion that involves combining several text processing components so that the output of one becomes the input for the next. The pipelines are an excellent and simple technique to infer from models. They are objects that abstract most of the library's complicated functionality, providing a straightforward API for a variety of tasks.

4.3 Implementation phase

In this section, we will explain the steps we took to build and implement our systems, as well as the various functions and parameters we employed.

4.3.1 Dataset description

XLSum [111] is a large and diverse dataset from the BBC that contains 1.35 million properly annotated article-summary pairs, which was gathered from news in 45 languages ranging from low-resource such as Bengali and Swahili languages to high-resource including English and Russian languages. This is accomplished by the crawler software, which recursively crawls pages of the BBC website beginning with the homepage and accessing various article links available on each page view, making use of the fact that all BBC websites have fairly similar architecture, as well as the provider of a full article summary in the form of a bold paragraph, which is produced professionally by the article's writers.

The process crawler gathers article summaries using a set of heuristics to ensure that the extraction is as efficient as possible where the required summary must appear in the first two paragraphs of the article. Also, a portion of the content in the summary paragraph must be bold, with the percentage of bold and hyperlinked texts equaling at least 95% of the overall length of the paragraph in question. In addition to the input text, must include all texts except the summary and the headline, and it must be at least twice as large as the summary. Any sample that did not meet these heuristics was thrown away. Human and intrinsic evaluations were performed on the dataset to guarantee that it is valuable and could be used by a larger community for summarization. We only used the Arabic language from this dataset as needed in our system, which has a total of 46897 Arabic rows.

4.3.2 Arabic text summarizer based on BART

This section concentrates on the specific implementation steps for text summarization utilizing two models, one based on Bart and the other on PEGASUS.

4.3.2.1 Loading and preprocessing the dataset

The dataset to be used is available as a dictionary. We use load_dataset to load it, and AutoTokenizer to preprocess it so that it can be accepted by the model. Before we can use these two methods, we must first install the datasets library and transformers in the Kaggle environment because they are not integrated into it previously.

```
! pip install transformers
! pip install datasets
```

Figure 4.6: Install datasets library and transformers.

After its installation, we import our requirements from them and use it.

```
from datasets import load_dataset

dataset = load_dataset("csebuetnlp/xlsum","arabic")
```

Figure 4.7: Import load_dataset from datasets.

The dataset is loaded. As we can see in Figure 4.8, it is in the form of a dictionary that contains three parts. There are 37519 rows in the training section (80%), 4689 in the validation section (10%), and 4689 rows in the test section (10%).

```
DatasetDict({
    train: Dataset({
        features: ['id', 'url', 'title', 'summary', 'text'],
        num_rows: 37519
    })
    test: Dataset({
        features: ['id', 'url', 'title', 'summary', 'text'],
        num_rows: 4689
    })
    validation: Dataset({
        features: ['id', 'url', 'title', 'summary', 'text'],
        num_rows: 4689
    })
})
```

Figure 4.8: Dataset description.

Data fields

- **id:** the article ID is represented as a string.
- **url:** is a string that represents the URL of an article.
- **title:** is a string that contains the title of the article.
- **summary:** is a string that contains the summary of the article.

- **text:** text is a string that contains the text of the article.

This is a sample of what the data looks like from the first line of the training dataset.

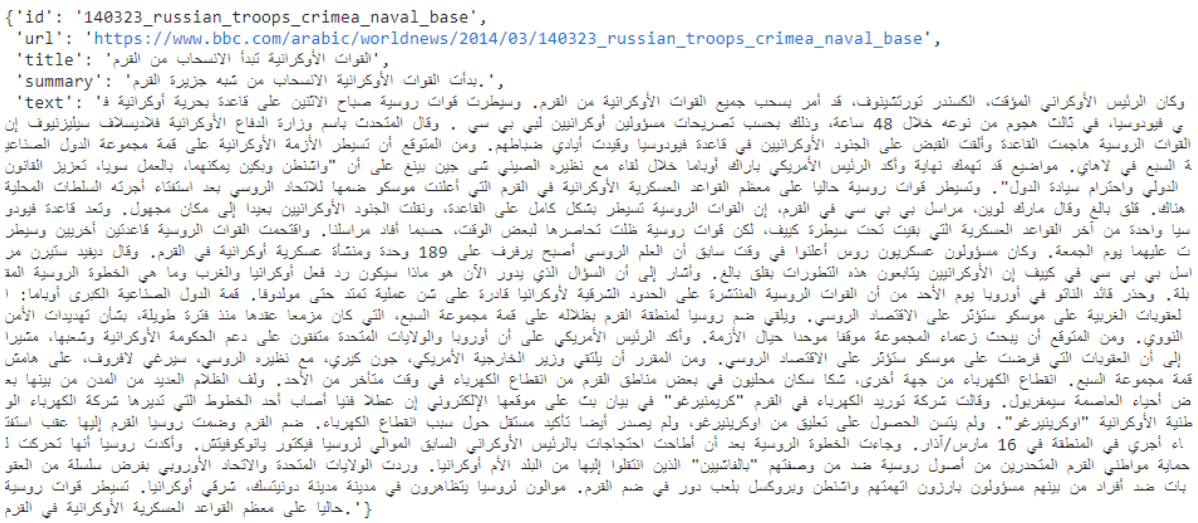

Figure 4.9: A representative sample of the data.

The data has already been processed, so there is no need for it. We only need to convert it to numerical values i.e. tokenize it to feed it to the pre-trained model. This is accomplished using a Transformers Tokenizer, which tokenizes inputs by breaking text into tokens and converting tokens to their matching IDs in the pre-trained vocabulary.

First, we have to load the associated pre-trained tokenizer of the model to guarantee that we obtain a tokenizer that matches the model architecture we're using, create the rest of the model's required inputs, get the vocabulary that was used during pretraining this specific checkpoint, divide the text similar to the pretraining corpus was divided.

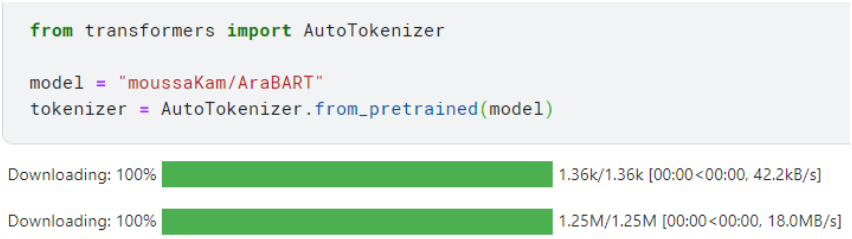

Figure 4.10: Load the associated pre-trained tokenizer of the model.

Second, we build a function that that will preprocess our data. We simply pass them to the tokenizer with the *truncation=True* option. This ensures that inputs that are longer than the model's maximum length will be truncated. We pad instances to the longest length in the batch rather than the entire dataset because padding will be handled later

(in a data collator). To ensure that the tokenization of inputs and labels is parallelized, we employed the *as_target_tokenizer()* context manager.

```python
max_input_length = 1024
max_target_length = 128

def preprocess_function(rows):
    inputs = [row for row in rows["text"]]
    model_inputs = tokenizer(inputs, max_length=max_input_length, truncation=True)

    # Setup the tokenizer for targets
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(rows["summary"], max_length=max_target_length, truncation=True)

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

Figure 4.11: The function that preprocesses the data.

Then, we pass a row of data to the tokenizer to see how it looks:

```python
preprocess_function(dataset['train'][:1])
```


```
{'input_ids': [[0, 193, 164, 26255, 11327, 4, 38570, 8401, 34042, 1473, 4, 78, 560, 16451, 130, 675, 24051, 6, 18399, 7, 38007, 622, 11509, 885, 822, 8, 2051, 6496, 28, 32849, 33, 5, 5,
26680, 10640, 4, 5, 4960, 2722, 6, 9637, 66, 2872, 782, 4, 160, 1051, 1681, 7309, 28, 32849, 496, 44, 1119, 473, 192, 14, 122, 1922, 495, 234, 561, 24051, 490, 1559, 19284, 192, 22153,
161, 1473, 72, 675, 1780, 30145, 2322, 37921, 1998, 8, 4807, 26255, 29, 5, 2051, 5, 26680, 10640, 10, 20845, 30, 38516, 6103, 39, 7, 103, 2287, 11, 8653, 1327, 24051, 8, 2587, 276, 279,
1697, 6170, 5, 34848, 7, 1745, 78, 35701, 497, 708, 164, 984, 6075, 3092, 66, 960, 26, 4043, 4642, 1644, 63, 8292, 56, 2062, 8, 11, 16, 16284, 10, 40638, 136, 356, 4, 2929, 14324, 4, 14
59, 429, 297, 5979, 7572, 279, 189, 149, 19907, 622, 11509, 635, 8, 946, 5022, 845, 24051, 5, 18399, 23, 929, 3257, 3722, 17, 3829, 1804, 51, 9236, 25377, 995, 697, 145, 7, 5903, 5252,
122, 8324, 311, 29, 4, 4215, 473, 473, 192, 5, 18399, 4, 72, 675, 1780, 8653, 148, 750, 8, 2322, 4, 4256, 4807, 26255, 29, 1905, 19, 493, 10653, 7, 5717, 2051, 5, 26680, 10640, 404, 6,
101, 5022, 845, 23, 8577, 210, 5075, 24340, 4, 158, 622, 11509, 7942, 38064, 17, 2543, 220, 4, 7647, 5734, 4215, 49, 7, 230, 25974, 30, 675, 1780, 17770, 285, 9062, 29, 38007, 7275, 10
7, 503, 7, 193, 8885, 3742, 45, 11624, 36062, 5, 263, 1630, 11, 732, 1804, 1024, 44734, 4966, 8, 31920, 1219, 10, 41138, 2015, 28, 32849, 33, 5, 18399, 7, 122, 6662, 292, 25648, 4215, 4
73, 473, 192, 5, 24340, 72, 26255, 29, 41526, 43, 4677, 41351, 5252, 7, 913, 19, 11, 2234, 41, 6098, 204, 62, 1526, 592, 527, 1193, 7122, 11616, 206, 92, 3432, 1780, 1097, 7, 10968, 153
6, 16177, 5, 1147, 107, 761, 6, 11, 675, 1780, 9953, 8, 1020, 888, 44, 27138, 3809, 8, 3787, 277, 7060, 80, 9232, 7951, 7, 2587, 279, 1697, 1170, 3092, 21, 2721, 999, 8, 3257, 49661, 8,
972, 1804, 7, 47395, 3722, 864, 9697, 18399, 35, 42944, 9, 8, 2587, 276, 6170, 4, 23, 59, 6902, 15880, 17704, 152, 534, 1048, 4, 701, 11970, 377, 2534, 7, 103, 2287, 11, 3144, 12593, 10
76, 15019, 10618, 12, 4958, 1327, 7, 708, 164, 984, 8, 11, 1147, 7079, 157, 10303, 45, 8, 683, 197, 24051, 28792, 4, 654, 19, 11, 2721, 23, 11375, 8, 3257, 49661, 8, 972, 1804, 7, 103,
2060, 11, 5229, 207, 368, 984, 4, 3110, 12396, 4, 26, 4043, 1804, 4, 19845, 14642, 4, 8, 4036, 2587, 276, 6170, 7, 7205, 1864, 6, 779, 181, 4, 27439, 1535, 9349, 45, 5, 104, 857, 18399,
6, 7205, 1864, 5, 263, 15683, 6, 761, 7, 10, 6817, 9850, 293, 6, 1673, 6, 1733, 104, 6807, 771, 18355, 2772, 3412, 7, 712, 135, 12622, 1864, 5, 18399, 16, 18449, 161, 36848, 42, 5, 613,
2231, 8, 6631, 775, 72, 14555, 12, 12889, 7411, 177, 3811, 23, 35836, 135, 1864, 342, 24051, 16, 12, 16147, 5392, 36848, 189, 241, 34728, 474, 8, 2090, 6, 127, 50, 35511, 36848, 4, 241,
3830, 155, 3405, 6229, 146, 1096, 7205, 1864, 7, 3722, 18399, 31891, 864, 18399, 807, 1167, 9236, 23313, 5, 242, 5, 547, 569, 3, 1299, 15800, 7, 6609, 3432, 1780, 51, 11, 37599, 11837,
17669, 26255, 410, 31823, 23465, 22783, 198, 4706, 42667, 7, 3760, 864, 301, 31489, 3527, 12904, 18399, 21188, 7233, 6, 3545, 11509, 259, 6, 12309, 39, 16, 2491, 423, 7192, 29, 42, 147,
4505, 53, 807, 6, 1153, 1642, 7122, 7, 7295, 382, 157, 5942, 1245, 15977, 1593, 6, 2721, 259, 1639, 6, 1157, 8885, 10465, 45, 16974, 39, 1372, 10, 32603, 5811, 39081, 441, 5, 3722, 1839
9, 7, 24597, 45, 23465, 46995, 5, 209, 209, 153, 14915, 50, 4, 7065, 7122, 7, 8653, 622, 11509, 635, 8, 946, 5022, 845, 24051, 5, 18399, 7, 2]], 'attention_mask': [[1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]], 'labels': [[0, 832, 675, 24051, 8410, 6, 2515, 3407, 18399, 7, 2]]}
```


Figure 4.12: Example of tokenizing the first row of data.

The tokenizer returns a dictionary that has three key items:

- **input_ids:** are indexes for each token in the text.
- **attention_mask:** determines whether or not a token should be attended to.
- **labels:** are the indices corresponding to each token in the summary.

Now, we pass the whole data to the tokenizer.

```
tokenized_dataset = dataset.map(preprocess_function, batched=True)
```

```
100% 38/38 [00:58<00:00, 1.21s/ba]
100% 5/5 [00:06<00:00, 1.16s/ba]
100% 5/5 [00:06<00:00, 1.17s/ba]
```

Figure 4.13: Tokenize the entire data.

After we tokenize the entire data, we feed it to the pre-trained model we intend to use.

4.3.2.2 Fine-tuning the model

The data has been preprocessed and is now ready to be fed to the chosen model. We chose AraBART, the pre-trained model, to fine-tune it on the XLSum dataset for the abstractive summarization task. We feed this model with preprocessed training and validation data to train it on this task.
First of all, we load the model that we have chosen to use.

```
from transformers import AutoModelForSeq2SeqLM

pretrained_model = AutoModelForSeq2SeqLM.from_pretrained(model)
```

Figure 4.14: Load the chosen model.

Trainer API

We will fine-tune the said model using the Trainer. The Trainer class in PyTorch provides an API for feature-complete training for the majority of standard use cases. Most of the example scripts make use of it. The API enables distributed training on multiple GPUs/TPUs, mixed-precision with NVIDIA Apex, and PyTorch Native AMP.

To access all of the points of customization during training, it is necessary to create TrainingArguments before creating a Trainer instance,which is a class that contains all the attributes to customize the training. Because our objective is to summarise, it is obvious that the input and output are both text sequences, i.e. sequence to sequence. In this situation, we use a Seq2SeqTrainer and Seq2SeqTrainingArguments.

To go through our training using the Trainer, we need three main things, the class that holds all of the variables that can be used to customize the training which is Seq2SeqTrainingArguments, a special kind of data collator called DataCollatorForSeq2Seq, and how to calculate the metrics based on the predictions. We simply need to pass all of this information after we've configured them, together with the preprocessed datasets, to the Seq2SeqTrainer. Figure 4.15 shows the import of the said things to use them.

```
from transformers import DataCollatorForSeq2Seq
from transformers import Seq2SeqTrainingArguments
from transformers import Seq2SeqTrainer
```

Figure 4.15: Import the necessities.

Now, we fill the class that contains all the attributes to customize the training. All remaining arguments are optional, except for the folder name, which will be used to save the model's checkpoints.

```
batch_size = 4
model_name = model.split("/")[-1]
arguments = Seq2SeqTrainingArguments(
    f"{model_name}-fine-xlsum",
    evaluation_strategy = "epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    predict_with_generate=True,
    #fp16=True
)
```

Figure 4.16: Seq2SeqTrainingArguments parameters.

The trainer needs a special kind of data collator. Data collators are objects that take a list of dataset elements as input and create a batch of data. These items have the same type as train_dataset and eval_dataset elements. Data collators may perform some processing in order to create batches (like padding). Some of them (such as DataCollatorForLanguageModeling) also do random data augmentation (such as random masking) on the resulting batch. In our condition, the DataCollatorForSeq2Seq will dynamically pad not just the inputs but also the labels to the batch's maximum length.

```
data_collator = DataCollatorForSeq2Seq(tokenizer, model=pretrained_model)
```

Figure 4.17: Defining the data collator.

Finally, we must give all of this information, as well as our datasets and the metric, to the Seq2SeqTrainer.

```
trainer = Seq2SeqTrainer(
    pretrained_model,
    arguments,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
```

Figure 4.18: Defining the trainer.

At the end, by simply executing the *train()* method, we can now fine-tune our model:

```
trainer.train()
```

Figure 4.19: Trainer fine-tune.

After 3 epochs of training we get these measures of ROUGE metric.

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	RougeI	RougeIsum	Gen Len
1	2.782200	2.379432	1.871800	0.056900	1.882800	1.876800	19.658300
2	2.499000	2.340080	1.920100	0.085300	1.929700	1.931500	19.614400
3	2.223600	2.339413	1.792700	0.109500	1.798700	1.794900	19.618700

Figure 4.20: Measurements of 3 epochs.

4.3.2.3 Prediction

The Trainer provides a method for testing the model on a test dataset termed *Predict*. It generates predictions on a test set (along with metrics if labels are available). So, we make the prediction.

```
outputs = trainer.predict(tokenized_dataset["test"])
```

Figure 4.21: Apply the prediction function on the test dataset.

Then, we decode the tokenized outputs to retrieve the generated summaries.

```
output_str = tokenizer.batch_decode(output.predictions, skip_special_tokens=True)
```

Figure 4.22: Decode the predictions to retrieve the summaries.

This is a samples of the generated summaries by the fine-tuned model.

```
['أصدرت محكمة عسكرية في ولاية فروت هود الأمريكية حكما بالإعدام على رجل قتل 13 جنديا',
 'وافق الائتلاف الحاكم في ألمانيا على السماح لمواطني خمس دول عربية بدخول البلاد بصورة غير قانونية.',
 'قال الجيش السوري إنه استعاد السيطرة على حي الخالدية في مدينة حمص من أيدي المعارضة المسلحة.',
```

Figure 4.23: Sample of the generated summaries.

4.3.3 Arabic text summarizer based on PEGASUS

Below we describe the steps involved in implementing an abstractive pegasus-based summarizer. First of all, we install the requirements so that we can use the various packages.

```
! pip install transformers
! pip install sentencepiece
! pip install tqdm
! pip install datasets
! pip install -U deep_translator
! pip install Rouge
```

Figure 4.24: Install requirements.

Second, the import of various libraries and packages that are necessary to implement and facilitate the process of building the system.

```
from transformers import pipeline
from tqdm import tqdm
from datasets import load_dataset
from deep_translator import GoogleTranslator
from rouge import Rouge
```

Figure 4.25: Import necessary packages and libraries.

After installing and importing the essentials, we define a pipeline with the task of *summarization* and the model to be used to perform this task. We chose *google/pegasus-xsum* for it.

```
Summarizer = pipeline("summarization", model="google/pegasus-xsum")
```

Figure 4.26: Define the summarizer.

The summarization tool is ready. The dataset remains that we have not loaded yet. We carry it now.

```
dataset = load_dataset("csebuetnlp/xlsum","arabic",split="test")
```

Figure 4.27: Load the dataset.

Before we can pass the data to the summarizer, we first have to translate it from Arabic to English because it does not support the Arabic language. We can try it with a simple Arabic text and see the result.

```
test_text = """
قال التلفزيون الرسمي السوري إن القوات الحكومية سيطرت على البلدة بالكامل ويأتي ذلك بعد أيام من تمكن القوات الحكومية السورية من طرد مسلحي التنظيم
من مدينة تدمر التاريخية المجاورة. وكان مسلحو تنظيم الدولة الاسلامية استولوا على بلدة القريتين في آب/أغسطس، واحتجزوا المئات من سكانها،
بينهم عشرات المسيحيين، وحرر العديد منهم لاحقا. وقالت جماعة حقوقية إن ثمة بعض جيوب القتال في البلدة. وكان المرصد السوري لحقوق الإنسان، ومقره بريطانيا
"""
test_summary = Summarizer(test_text)
test_summary
```

```
[{'summary_text': '      800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-
3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 80
0-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3
217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800
-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-32
17 800-273-3217 800-273-3217 800-273-3217 800-273-3217 800-273-3217'}]
```

Figure 4.28: Test pegasus aptitude of Arabic.

In Figure 4.28, we defined a text in Arabic and passed it to the summarizer to prove that pegasus doesn't accept Arabic. So in Figure 4.29, we have translated the input into the language it accepts, which is English.

```
english_text = []
pbar = tqdm(total=3, position=0, leave=True)

for i in range(0,3):
    english_text.append(GoogleTranslator(source='ar', target='en').translate(dataset["text"][i][0:4999]))
    pbar.update(1)
pbar.close()
```

```
100%|██████████| 3/3 [00:00<00:00,  3.09it/s]
```

Figure 4.29: Translation of texts to English.

For the translation process and all subsequent operations, we applied them to only the first three texts of the dataset to reduce the computations.

```
dataset["text"][0]
```

```
'نضال حسن واعترف نضال حسن، الذي يدافع عن نفسه، بقتل الجنود، متحججا بحماية المسلمين وعناصر طالبان في أفغانستان، ولكن القاضي العسكري رفض حجته "بحماية الآخرين". وإذا أدين حسن،
البالغ من العمر 42 عاما، بقتل 13 شخصا وجرح آخرين فإنه سيواجه عقوبة الإعدام. ويعتبر الحادث الأكثر دموية من بين الهجمات غير القتالية التي وقعت في قاعدة عسكرية أمريكية. وقال شهود عيا
ن دخل في 5 نوفمبر/تشرين الثاني عام 2009 مصحة تعج بالجنود الذين كانوا ينتظرون أدوارهم إجراء فحوصات طبية أو التلقيح، ثم صعد على مكتب، وأطلق النار من سلاحين بيديه، دون توقف إلا
لإعادة تعبئة السلاح. مواضيع قد تهمك نهاية وسيقدم ممثلو الادعاء أدلة تفيد بأن حسن مال إلى الأفكار المتطرفة، وكان يزور المواقع بحثا عن"الجهاديين" وطالبان، ساعات قبل الهجوم. وكان الرائد حسـ
ن سيلتحق بالقوات الأمريكية في أفغانستان قبل أن ينفذ هجومه. "عنف في مكان العمل" وصنفت وزارة الدفاع الأمريكية الحادث باعتباره "عنفا في مكان العمل" بدلا من تصنيفه "عملا إرهابيا"، وهو
ما أغضب عائلات الضحايا، حسب ما أفاد به مراسل بي بي سي، تك برايت، في فروت هود. ويتوقع أن يدلي العديد من جرحى الحادث بشاهاداتهم أمام المحكمة. وسيواجه حسن عددا من ضحاياه في
قاعة المحكمة لأنه سيتولى الدفاع عن نفسه. وهو يستخدم كرسيا متحركا لأنه أصيب بالشلل، عندما أطلق عليه شرطي في القاعدة العسكرية النار.'
```

Figure 4.30: The original text.

Figures 4.30 and 4.31 represent text in its original language and after translation, respectively.

```
english_text[0]
```

```
'Nidal Hassan Nidal Hassan, who was defending himself, admitted to killing the soldiers, citing protection of Muslims and Taliban
elements in Afghanistan, but the military judge rejected his argument for "protecting others". If Hassan, 42, is convicted of kil
ling 13 people and wounding others, he will face the death penalty. The incident is considered the deadliest among the non-combat
attacks that took place on a US military base. On November 5, 2009, eyewitnesses said, he entered a clinic full of soldiers waiti
ng for their turn to undergo medical examinations or vaccinations, then climbed onto an office and fired two weapons in his hand
s, without stopping except to reload the weapon. Topics that may interest you At the end, prosecutors will present evidence that
Hasan ran into extremist ideas, and was visiting sites looking for jihadists and the Taliban, hours before the attack. Major Hass
an would have joined US forces in Afghanistan before he carried out his attack. The US defense department described the incident
as "workplace violence" rather than classifying it as a "terrorist act", which infuriated the families of the victims, BBC corres
pondent Nick Bryant reported in Fruit Hood. Many of those injured are expected to testify in court. Hassan will face a number of
his victims in the courtroom because he will defend himself.He uses a wheelchair because he was paralyzed when a policeman at the
military base shot him.'
```

Figure 4.31: The translated text.

In Figure 4.32, we pass the translated text through the summarizer.

```
summary = []
for txt in english_text:
    summary.append(Summarizer(txt)[0]['summary_text'])
```

Figure 4.32: Summarize the translated texts.

Figure 4.33 illustrates the given summaries were generated by pegasus. Next, we will translate it back into Arabic.

```
summary
```

```
['The trial of a US Army psychiatrist accused of killing 13 people at a Texas military base in 2009 has begun.',
 "Germany's governing coalition has agreed to send home the families of migrants from Syria, Iraq and Morocco.",
 'The Syrian army says it has taken full control of the Khalidiyah district in the central city of Homs.']
```

Figure 4.33: The summaries of the translated texts

In this Figure (4.34), we feed the English summaries to the translator to get the Arabic version of them.

```
pegasus_summary = []
pbar = tqdm(total=3, position=0, leave=True)

for summ in summary:
    pegasus_summary.append(GoogleTranslator(source='en', target='ar').translate(summ[0:4999]))
    pbar.update(1)
pbar.close()
```

```
100%|██████████| 3/3 [00:00<00:00,  3.82it/s]
```

Figure 4.34: Translation of summaries to Arabic.

Our final Arabic summaries look like this.

```
pegasus_summary
```

```
['2009 بدأت محاكمة طبيب نفسي بالجيش الأمريكي متهم بقتل 13 شخصًا في قاعدة عسكرية بتكساس عام.',
 'وافق الائتلاف الحاكم في ألمانيا على إعادة عائلات المهاجرين من سوريا والعراق والمغرب.',
 'أعلن الجيش السوري عن سيطرته الكاملة على حي الخالدية وسط مدينة حمص.']
```

Figure 4.35: The final form of the summaries.

4.4 Comparison and discussion

In this section, we are going to evaluate the summaries resulting from our text summarizers using the metric ROUGE and its variants. We are interested only in the stat F-measure for three metrics which are ROUGE-1, ROUGE-2, ROUGE-L.

In Figure 4.36, We fetched the first three abstracts generated with the first AraBART-based summarizer, whose texts are equivalent to the three used in the Pegasus-based summarizer, and gave them the name *model_summary*. Then, we computed the scores of these summaries compared to the original summary contained in the dataset.

```
scores_model_label = rouge.get_scores(model_summary,dataset["summary"][:3])
for i in range (0,3):
    print("rouge-1 : ",scores_model_label[i]['rouge-1']['f']*100)
    print("rouge-2 : ",scores_model_label[i]['rouge-2']['f']*100)
    print("rouge-L : ",scores_model_label[i]['rouge-l']['f']*100)
    print("")
```

```
rouge-1 :  20.83333290364584
rouge-2 :  4.0816322448980005
rouge-L :  16.66666623697918

rouge-1 :  10.52631530193908
rouge-2 :  0.0
rouge-L :  5.263157407202262

rouge-1 :  37.8378373469686
rouge-2 :  11.42857093877553
rouge-L :  37.8378373469686
```

Figure 4.36: Scores of the AraBART-based summarizer.

In Figure 4.37, we did the same thing to the summaries generated by the pegasus-based summarizer, in which the computation of scores compared to the original summary contained in the dataset.

```
scores_pegasus_label = rouge.get_scores(pegasus_summary,dataset["summary"][:3])
for i in range (0,3):
    print("rouge-1 : ",scores_pegasus_label[i]['rouge-1']['f']*100)
    print("rouge-2 : ",scores_pegasus_label[i]['rouge-2']['f']*100)
    print("rouge-L : ",scores_pegasus_label[i]['rouge-l']['f']*100)
    print("")
```

```
rouge-1 :  16.326530172428168
rouge-2 :  3.9999995800000443
rouge-L :  12.244897519366944

rouge-1 :  28.571428104489797
rouge-2 :  0.0
rouge-L :  11.428570961632674

rouge-1 :  36.363635900826445
rouge-2 :  19.35483825182103
rouge-L :  36.363635900826445
```

Figure 4.37: Scores of Pegasus-based summarizer.

Finally, we calculated the scores between the summaries generated by the summarization systems and the results are shown in Figure 4.38

```
scores_model_pegasus = rouge.get_scores(model_summary,pegasus_summary)
for i in range (0,3):
    print("rouge-1 : ",scores_model_pegasus[i]['rouge-1']['f']*100)
    print("rouge-2 : ",scores_model_pegasus[i]['rouge-2']['f']*100)
    print("rouge-L : ",scores_model_pegasus[i]['rouge-l']['f']*100)
    print("")
```

```
rouge-1 :  19.354838210197727
rouge-2 :  0.0
rouge-L :  6.451612403746138

rouge-1 :  41.37930985017836
rouge-2 :  37.03703654320989
rouge-L :  41.37930985017836

rouge-1 :  49.99999951020409
rouge-2 :  30.76923028106509
rouge-L :  49.99999951020409
```

Figure 4.38: Scores calculated between AraBART-based and Pegasus-based summarizers.

To connect the points, we summarized our results in Table 4.1 to be clear and legible.

	Rouge-1	Rouge-2	Rouge-L	
AraBART-based	**20.83**	**4.08**	**16.66**	Summary 1
	10.52	0.0	5.26	Summary 2
	37.83	11.42	37.83	Summary 3
Pegasus-based	16.32	3.99	12.24	Summary 1
	28.57	**0.0**	**11.42**	Summary 2
	36.36	**19.35**	**36.36**	Summary 3
AraBART-based vs Pegasus-based	19.35	0.0	6.45	Summary 1
	41.37	**37.03**	**41.37**	Summary 2
	49.99	**30.76**	**49.99**	Summary 3

Table 4.1: Evaluation of text summarizers.

According to the results obtained, Text 1 has the highest scores Rouge-1, Rouge-2 and Rouge-L from the AraBART-based summarizer. As for the second text, the high scores

were from the summarizer based on Pegasus. For the third text, the results are mixed. It scored a slightly higher percentage from the first system (AraBART-based summarizer) than the ones that scored from the second in both Rouge-1 and Rouge-l, and in Rouge-2 it got a high percentage from the summarizer based on Pegasus.

From the last part of the table that compares the summaries generated from the two systems, we notice that the abstract summaries produced by these summarizers are close. The high degree of similarity in the vocabulary used increased to 49.99% in the summary of the third text, and 41.37% in the summary of the second text.

4.5 Conclusion

This chapter discusses the context in which the programs are developed, which includes implementation and experiments. In addition, the key results of the theme were discussed and explained, as well as comparisons of the acquired results through a table with annotations.

The results obtained indicate a rather acceptable performance, allowing us to continue to improve it.

Conclusion and future work

This study is useful in getting to know one of the most well-known tasks that belong to the broad field of natural language processing and identifying the latest methods used to give words representations that help in understanding their intended meaning. The task of automatic summarization is a task that received a lot of attention, especially after the emergence of contextualized word embedding that helped very much in creating summaries similar to those created by humans.

In this work, we have studied several pre-trained models and proposed two approaches to summarize the Arabic text based on it. The first is based on BART's model and the other on Pegasus. We designed, implemented and compared their results. In general, our summarizers produce similar and somewhat acceptable results despite the difficulties of Arabic that encounter such work. However, due to hardware limitations and unavailability of the necessary environment, we were unable to train our models for extended epochs, nor did we test our models on other data sets to determine if they were effective.

In the future, we will try to continue the path of improving and enhancing the results of the AraBART-based model, fine-tuning our model on a larger dataset, and training it on the task of summarizing Arabic reviews.

Bibliography

[1] Hans Peter Luhn. "The automatic creation of literature abstracts". In: *IBM Journal of research and development* 2.2 (1958), pp. 159–165.

[2] Asma Bader Al-Saleh and Mohamed El Bachir Menai. "Automatic Arabic text summarization: a survey". In: *Artificial Intelligence Review* 45.2 (2016), pp. 204, 205.

[3] Lamees Mahmoud Al Qassem et al. "Automatic Arabic summarization: a survey of methodologies and systems". In: *Procedia Computer Science* 117 (2017), p. 11.

[4] Diksha Khurana et al. "Natural language processing: State of the art, current trends and challenges". In: *arXiv preprint arXiv:1708.05148* (2017), pp. 6, 9.

[5] Bing Liu. "Sentiment analysis and opinion mining". In: *Synthesis lectures on human language technologies* 5.1 (2012), p. 7.

[6] Zara Nasar, Syed Waqar Jaffry, and Muhammad Kamran Malik. "Textual keyword extraction and summarization: State-of-the-art". In: *Information Processing & Management* 56.6 (2019), pp. 17, 19.

[7] *Summary*. URL: `https://dictionary.cambridge.org/dictionary/english/%20Summary`.

[8] Karen Sparck Jones. "Automatic summarising: factors and directions". In: *arXiv e-prints* (1998), p. 2.

[9] Dragomir Radev, Eduard Hovy, and Kathleen McKeown. "Introduction to the special issue on summarization". In: *Computational linguistics* 28.4 (2002), pp. 399–408.

[10] Mohamed Hedi Maaloul. "Approche hybride pour le résumé automatique de textes. Application à la langue arabe." PhD thesis. Université de Provence-Aix-Marseille I, 2012, pp. 9, 10.

[11] Mani Maybury. *Advances in automatic text summarization*. MIT press, 1999.

[12] Elena Lloret. "Text summarization: an overview". In: *Paper supported by the Spanish Government under the project TEXT-MESS (TIN2006-15265-C06-01)* (2008).

[13] Mahmood Yousefi-Azar and Len Hamey. "Text summarization using unsupervised deep learning". In: *Expert Systems with Applications* 68 (2017), pp. 93–105.

[14] Wafaa S El-Kassas et al. "Automatic text summarization: A comprehensive survey". In: *Expert Systems with Applications* 165 (2021), pp. 4, 5.

[15] Monika Joshi, Hui Wang, and Sally McClean. "Dense semantic graph and its application in single document summarisation". In: *Emerging Ideas on Information Filtering and Retrieval*. Springer, 2018, p. 56.

[16] Mahak Gambhir and Vishal Gupta. "Recent automatic text summarization techniques: a survey". In: *Artificial Intelligence Review* 47.1 (2017), pp. 3, 39.

[17] Saeedeh Gholamrezazadeh, Mohsen Amini Salehi, and Bahareh Gholamzadeh. "A comprehensive survey on text summarization systems". In: *2009 2nd International Conference on Computer Science and its Applications*. IEEE. 2009, p. 2.

[18] Eduard Hovy, Chin-Yew Lin, et al. "Automated text summarization in SUMMARIST". In: *Advances in automatic text summarization* 14 (1999), p. 197.

[19] Vishal Gupta and Narvinder Kaur. "A novel hybrid text summarization system for Punjabi text". In: *Cognitive Computation* 8.2 (2016), p. 261.

[20] Bilel Elayeb et al. "Automatic arabic text summarization using analogical proportions". In: *Cognitive Computation* 12.5 (2020), p. 1.

[21] Neelima Bhatia and Arunima Jaiswal. "Trends in extractive and abstractive techniques in text summarization". In: *International Journal of Computer Applications* 117.6 (2015).

[22] Vishal Gupta and Gurpreet Singh Lehal. "A survey of text summarization extractive techniques". In: *Journal of emerging technologies in web intelligence* 2.3 (2010), pp. 258, 259.

[23] Liwei Hou, Po Hu, and Chao Bei. "Abstractive document summarization via neural model with joint attention". In: *National CCF Conference on Natural Language Processing and Chinese Computing*. Springer. 2017, pp. 1, 2, 3.

[24] Nikita Munot and Sharvari S Govilkar. "Comparative study of text summarization methods". In: *International Journal of Computer Applications* 102.12 (2014), pp. 34, 35.

[25] Amol Tandel et al. "Multi-document text summarization-a survey". In: *2016 International Conference on Data Mining and Advanced Computing (SAPIENCE)*. IEEE. 2016, p. 1.

[26] NR Kasture et al. “A survey on methods of abstractive text summarization”. In: *Int. J. Res. Merg. Sci. Technol* 1.6 (2014), p. 54.

[27] Aziz Qaroush et al. “An efficient single document Arabic text summarization using a combination of statistical and semantic features”. In: *Journal of King Saud University-Computer and Information Sciences* 33.6 (2021), p. 678.

[28] N Moratanch and S Chitrakala. “A survey on abstractive text summarization”. In: *2016 International Conference on Circuit, power and computing technologies (ICCPCT)*. IEEE. 2016, p. 1.

[29] S Chitrakala et al. “Concept-based extractive text summarization using graph modelling and weighted iterative ranking”. In: *International Conference on Emerging Research in Computing, Information, Communication and Applications*. Springer. 2016, p. 150.

[30] Rui Sun et al. “Query-biased multi-document abstractive summarization via submodular maximization using event guidance”. In: *International Conference on Web-Age Information Management*. Springer. 2016, p. 310.

[31] Pierre-Etienne Genest and Guy Lapalme. “Framework for abstractive summarization using text-to-text generation”. In: *Proceedings of the workshop on monolingual text-to-text generation*. 2011, p. 64.

[32] Natalya Shakhovska. *Advances in intelligent systems and computing*. Springer, 2017, p. 41.

[33] Shuai Wang et al. “Integrating extractive and abstractive models for long text summarization”. In: *2017 IEEE International Congress on Big Data (BigData Congress)*. IEEE. 2017, p. 305.

[34] Imed Zitouni. *Natural language processing of semitic languages*. Springer, 2014.

[35] Nizar Y Habash. “Introduction to Arabic natural language processing”. In: *Synthesis lectures on human language technologies* 3.1 (2010), pp. 1–187.

[36] Clive Holes. *Modern Arabic: Structures, functions, and varieties*. Georgetown University Press, 2004.

[37] Hani S AlGhanem and Rashan H Ajamiah. “Arabic text summarization approaches: A comparison study”. In: *International Journal of Information Technology and Language Studies* 4.3 (2020), p. 30.

[38] Fatima T Al-Khawaldeh. “Answer extraction for why Arabic questions answering systems: EWAQ”. In: *arXiv preprint arXiv:1907.04149* (2019), p. 83.

[39] Janice Shah et al. “Natural language processing based abstractive text summarization of reviews”. In: *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*. IEEE. 2020, pp. 461–466.

[40] Houda Oufaida, Omar Nouali, and Philippe Blache. "Minimum redundancy and maximum relevance for single and multi-document Arabic text summarization". In: *Journal of King Saud University-Computer and Information Sciences* 26.4 (2014), pp. 450–461.

[41] Hanchuan Peng, Fuhui Long, and Chris Ding. "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy". In: *IEEE Transactions on pattern analysis and machine intelligence* 27.8 (2005), pp. 1226–1238.

[42] Qasem A Al-Radaideh and Dareen Q Bataineh. "A hybrid approach for arabic text summarization using domain knowledge and genetic algorithms". In: *Cognitive Computation* 10.4 (2018), pp. 651–669.

[43] Abdullah M Abu Nada et al. "Arabic text summarization using arabert model using extractive text summarization approach". In: (2020).

[44] Abdullah Alshanqiti et al. "Leveraging DistilBERT for Summarizing Arabic Text: An Extractive Dual-Stage Approach". In: *IEEE Access* 9 (2021), pp. 135594–135607.

[45] Chin-Yew Lin. "Rouge: A package for automatic evaluation of summaries". In: *Text summarization branches out*. 2004, pp. 74–81.

[46] Rana Alqaisi, Wasel Ghanem, and Aziz Qaroush. "Extractive multi-document Arabic text summarization using evolutionary multi-objective optimization with K-medoid clustering". In: *IEEE Access* 8 (2020), p. 228218.

[47] Siwei Lai et al. "How to generate a good word embedding". In: *IEEE Intelligent Systems* 31.6 (2016), p. 5.

[48] S Selva Birunda and R Kanniga Devi. "A review on word embedding techniques for text classification". In: *Innovative Data Communication Technologies and Application* (2021), pp. 267–281.

[49] Hans Christian, Mikhael Pramodana Agus, and Derwin Suhartono. "Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF)". In: *ComTech: Computer, Mathematics and Engineering Applications* 7.4 (2016), pp. 285–294.

[50] Juan Ramos et al. "Using tf-idf to determine word relevance in document queries". In: *Proceedings of the first instructional conference on machine learning*. Vol. 242. 1. Citeseer. 2003, pp. 29–48.

[51] Abhinav Sethy and Bhuvana Ramabhadran. "Bag-of-word normalized n-gram models". In: *Ninth Annual Conference of the International Speech Communication Association*. 2008.

[52] B Oscar Deho et al. "Sentiment analysis with word embedding". In: *2018 IEEE 7th International Conference on Adaptive Science & Technology (ICAST)*. IEEE. 2018, p. 1.

[53] Felipe Almeida and Geraldo Xexéo. "Word embeddings: A survey". In: *arXiv preprint arXiv:1901.09069* (2019).

[54] V Kishore Ayyadevara. "Word2vec". In: *Pro Machine Learning Algorithms*. Springer, 2018, p. 1.

[55] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[56] Dongwen Zhang et al. "Chinese comments sentiment classification based on word2vec and SVMperf". In: *Expert Systems with Applications* 42.4 (2015), pp. 1857–1863.

[57] Bin Wang et al. "Evaluating word embedding models: Methods and experimental results". In: *APSIPA transactions on signal and information processing* 8 (2019).

[58] Piotr Bojanowski et al. "Enriching word vectors with subword information". In: *Transactions of the association for computational linguistics* 5 (2017), pp. 135–146.

[59] Armand Joulin et al. "Bag of tricks for efficient text classification". In: *arXiv preprint arXiv:1607.01759* (2016).

[60] Nils Reimers et al. "Classification and clustering of arguments with contextualized word embeddings". In: *arXiv preprint arXiv:1906.09821* (2019).

[61] Justyna Sarzynska-Wawer et al. "Detecting formal thought disorder by deep contextualized word representations". In: *Psychiatry Research* 304 (2021), p. 114135.

[62] Matej Ulčar and Marko Robnik-Šikonja. "High quality ELMo embeddings for seven less-resourced languages". In: *arXiv preprint arXiv:1911.10049* (2019).

[63] *bert*. URL: `https://github.com/google-research/bert`.

[64] Qihao Zhu and Jianxi Luo. "Generative pre-trained transformer for design concept generation: an exploration". In: *Proceedings of the Design Society* 2 (2022), pp. 3–4.

[65] Shirui Wang, Wenan Zhou, and Chao Jiang. "A survey of word embeddings based on deep learning". In: *Computing* 102.3 (2020), pp. 717–740.

[66] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. "Activation functions in neural networks". In: *towards data science* 6.12 (2017), p. 310.

[67] Yogesh Gupta and Amit Saraswat. "Machine Learning Techniques for Short-Term Forecasting of Wind Power Generation". In: *International Conference on Advanced Machine Learning Technologies and Applications*. Springer. 2020, p. 442.

[68] Uday Kamath, John Liu, and James Whitaker. *Deep learning for NLP and speech recognition*. Vol. 84. Springer, 2019, pp. 142, 155.

[69] Yoshua Bengio. *Learning deep architectures for AI*. Now Publishers Inc, 2009.

[70] WJ Zhang et al. "On definition of deep learning". In: *2018 World automation congress (WAC)*. IEEE. 2018, p. 234.

[71] Guillaume Chassagnon et al. "Deep learning: definition and perspectives for thoracic imaging". In: *European radiology* 30.4 (2020), p. 2022.

[72] Ping Li, Jianping Li, and Gongcheng Wang. "Application of convolutional neural network in natural language processing". In: *2018 15th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*. IEEE. 2018, pp. 120–122.

[73] Laith Alzubaidi et al. "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions". In: *Journal of big Data* 8.1 (2021), p. 13.

[74] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 international conference on engineering and technology (ICET)*. Ieee. 2017, pp. 1–6.

[75] Kyunghyun Cho, Aaron Courville, and Yoshua Bengio. "Describing multimedia content using attention-based encoder-decoder networks". In: *IEEE Transactions on Multimedia* 17.11 (2015), pp. 1875, 1876.

[76] Li Deng and Dong Yu. "Deep learning: methods and applications". In: *Foundations and trends in signal processing* 7.3–4 (2014), p. 220.

[77] Akshay Kulkarni and Adarsha Shivananda. *Natural language processing recipes*. Springer, 2019.

[78] Amitha Mathew, P Amudha, and S Sivakumari. "Deep learning techniques: an overview". In: *International Conference on Advanced machine learning technologies and applications*. Springer. 2020, p. 604.

[79] Yong Yu et al. "A review of recurrent neural networks: LSTM cells and network architectures". In: *Neural computation* 31.7 (2019), pp. 1236, 1238, 1239, 1241.

[80] Mathias Berglund et al. "Bidirectional recurrent neural networks as generative models". In: *Advances in neural information processing systems* 28 (2015), p. 3.

[81] Tehseen Zia and Usman Zahid. "Long short-term memory recurrent neural network architectures for Urdu acoustic modeling". In: *International Journal of Speech Technology* 22.1 (2019), pp. 21–30.

[82] Yonghui Wu et al. "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144* (2016).

[83] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[84] Kyunghyun Cho et al. "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259* (2014).

[85] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems* 27 (2014).

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

[87] William James et al. *The principles of psychology*. Vol. 1. 2. Macmillan London, 1890.

[88] Alana de Santana Correia and Esther Luna Colombini. "Attention, please! a survey of neural attention models in deep learning". In: *arXiv preprint arXiv:2103.16775* (2021), pp. 3, 13.

[89] Andrea Galassi, Marco Lippi, and Paolo Torroni. "Attention in natural language processing". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.10 (2020), pp. 4293, 4295.

[90] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[91] Dichao Hu. "An introductory survey on attention mechanisms in NLP problems". In: *Proceedings of SAI Intelligent Systems Conference*. Springer. 2019, p. 3.

[92] Thomas Wolf et al. "Huggingface's transformers: State-of-the-art natural language processing". In: *arXiv preprint arXiv:1910.03771* (2019).

[93] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[94] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[95] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

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

[97] Jacob Devlin and Ming-Wei Chang. "Open sourcing BERT: State-of-the-art pre-training for natural language processing". In: *Google AI Blog* 2 (2018).

[98] Anmol Nayak et al. "Domain adaptation challenges of BERT in tokenization and sub-word representations of Out-of-Vocabulary words". In: *Proceedings of the First Workshop on Insights from Negative Results in NLP*. 2020, pp. 1–5.

[99] Yinhan Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[100] Victor Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108* (2019).

[101] Wissam Antoun, Fady Baly, and Hazem Hajj. "Arabert: Transformer-based model for arabic language understanding". In: *arXiv preprint arXiv:2003.00104* (2020).

[102] *AraBert*. URL: https://github.com/aub-mind/arabert.

[103] Mike Lewis et al. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension". In: *arXiv preprint arXiv:1910.13461* (2019).

[104] Jingqing Zhang et al. "Pegasus: Pre-training with extracted gap-sentences for abstractive summarization". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 11328–11339.

[105] *PEGASUS: A State-of-the-Art Model for Abstractive Text Summarization*. URL: https://ai.googleblog.com/2020/06/pegasus-state-of-art-model-for.html.

[106] *python*. URL: https://www.python.org/.

[107] *kaggle*. URL: https://www.kaggle.com/.

[108] *PyTorch*. URL: https://pytorch.org/.

[109] *transformers*. URL: https://huggingface.co/docs/transformers/main/en/index.

[110] *Deep$_t$ranslator*. URL: https://github.com/nidhaloff/deep-translator.

[111] Tahmid Hasan et al. "XL-sum: Large-scale multilingual abstractive summarization for 44 languages". In: *arXiv preprint arXiv:2106.13822* (2021).