PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

YAHIA FARES UNIVERSITY OF MEDEA



FACULTY OF SCIENCES

DEPARTEMENT OF MATHEMATICS AND COMPUTER SCIENCES

Graduation Project in:

Computer Science Master

Speciality: Systems Engineering and Web Technology

A Deep Learning Model for an Arabic Text Summarization.

Proposed and Directed by:

- **Mr. HADJIR Ismail**

Prepared and Presented by:

- **BAKHTA Amina**
- **LADJALI Manel**

Academic Year : 2023/2024.

Acknowledgements:

First and foremost, we would like to express our gratitude to Allah for the blessings and achievements we have attained today. We extend our heartfelt appreciation to all those who have contributed directly and indirectly to the preparation of this study.

We also take this opportunity to acknowledge and thank our supervisor, Mr. HADJIR Ismail, for his unwavering attention, encouragement, and guidance throughout the duration of this study.

We are profoundly grateful to the president of the jury, Mr. TOUBAL Elbahi, and the jury member, Mr. HAYI Mohamed Yassine, for agreeing to evaluate this work.

Our heartfelt thanks go to our beloved family for their constant support and understanding.

Lastly, we are deeply thankful to all the teachers in the Faculty of Science at University Yahia Fares of Medea for their invaluable help and support at every stage of this research. Their expertise and encouragement have been instrumental in shaping this study.

Dedication:

I dedicate this project to:

God, for His endless blessings and guidance throughout my journey. Your divine presence has been my source of strength and inspiration.

My parents, whose unwavering support, love, and sacrifices have provided me with the opportunities and motivation to pursue my dreams. Your belief in me has been the cornerstone of my achievements.

My grandfather, for his wisdom, encouragement, and the values he has instilled in me. Your words and actions have always been a guiding light in my life.

To **the soul of my grandmother**, whose memory continues to inspire and motivate me. Though you are no longer with us, your spirit and love remain a constant source of comfort and encouragement.

My sisters, **Rima** and **Nada**, and my brother, **Rafik**, for their constant encouragement, understanding, and unwavering belief in my abilities. Your support has been instrumental in my success, and I am grateful for your presence in my life.

My project partner, **Amina**, for her collaboration, dedication, and hard work throughout this project. Your partnership has been invaluable, and I am grateful for the journey we have shared.

To all who supported me and helped me reach my goals, including friends, mentors, and colleagues. Your belief in me, encouragement, and guidance have been crucial in helping me navigate this journey and achieve my dreams. I am deeply thankful for each and every one of you.

LADJALI Manel.

Dedication:

First of all, I thank the **God**, who helped us to complete this modest work, I dedicate this work:

To my dear **parents**, Thank you for giving me the support to reach my dreams. Accomplishing this would hopefully make you proud of me as much as I am proud of having you as my parents. I love you, mama and papa.

To my sisters **Hanaa** and **Rahaf**, for their precious support, comforting presence, and love that enriched every moment of this adventure.

To my brothers **Haythem** and **Mohamed Amine**, for their steadfast support, unwavering camaraderie, and constant inspiration that fueled my determination.

To my twin sister **Wafaa**, for her invaluable presence, unwavering support, and inspiring strength that accompanied me at every step of this journey.

To my project partner **Manel**, for her invaluable collaboration, unwavering dedication, and exemplary partnership that contributed to the success of this project.

To my friends, for their loyal friendship, encouraging words, and moral support that sustained me through the toughest times.

To my teachers and mentors, for their invaluable guidance, expertise, and teachings that shaped my understanding and academic journey.

To all those who believed in me and helped me achieve my dreams, I humbly dedicate this work to you.

BAKHTA Amina.

Abstract:

Recently, with the availability of a vast amount of information and texts on the Internet, the comprehensive reading of long texts has become difficult and complicated. Based on this, summaries have become crucial for extracting the essence of textual documents. This study proposes a model for the automatic generation of summaries for Arabic texts based on the mBART deep learning model. Our methodology aims to produce summaries that synthesize the main ideas of the texts while maintaining coherence with the original text. The XL-Sum dataset for Arabic was used to train and evaluate our model, and its performance was also measured using ROUGE metrics, which assess the quality of the summaries by comparing the similarity between the generated summaries and human references. The results highlight the promising performance of our model, paving the way for future advancements in the field of automatic summarization of Arabic texts.

Keywords: Automatic text summarization, Abstractive summarization, Arabic language, mBART, Transformers, NLP, Deep Learning, XLSum.

Résumé:

Récemment, avec la disponibilité d'une énorme quantité d'informations et de textes sur Internet, la lecture complète de longs textes est devenue difficile et compliquée. Sur cette base, les résumés sont devenus cruciaux pour extraire l'essence des documents textuels. Cette étude propose un modèle pour la génération automatique de résumés de textes arabes basé sur le modèle d'apprentissage profond mBART. Notre méthodologie vise à produire des résumés qui synthétisent les idées principales des textes tout en maintenant une cohérence avec le texte original. L'ensemble de données XL-Sum pour l'arabe a été utilisé pour entraîner et évaluer notre modèle, et sa performance a également été mesurée à l'aide des métriques ROUGE, qui évaluent la qualité des résumés en comparant la similitude entre les résumés générés et les références humaines. Les résultats mettent en évidence les performances prometteuses de notre modèle, ce qui ouvre la porte à de futures avancées dans le domaine de la synthèse automatique des textes arabes.

mots-clés:

Résumé automatique de texte, Résumé abstrait, Langue arabe, mBART, Transformateurs, Traitement du langage naturel (TALN), Apprentissage profond, XLSUM.

ملخص:

مؤخرًا، في ظل تزايد الوصول إلى كميات كبيرة من المعلومات مما يجعل قراءة النصوص الطويلة بشكل شامل أمرًا صعبًا، أصبحت الملخصات أمرًا حاسمًا لاستخراج جوهر الوثائق. تقدم هذه الدراسة نموذجًا لملخصات النصوص باللغة العربية استنادًا إلى نموذج mBART. تهدف منهجيتنا إلى توليد ملخصات تلخص الأفكار الرئيسية مع الحفاظ على التماسك مع النص الأصلي. نستخدم مجموعة البيانات XL-Sum للعربية لتدريب وتقييم نموذجنا. يتم تقييم أدائه باستخدام مقاييس ROUGE، التي تقيم جودة الملخصات من خلال مقارنة التشابه بين الملخصات المولدة والمراجع البشرية. تبرز النتائج أداء واعدًا لنموذجنا، مما يفتح الباب أمام تقدمات مستقبلية في مجال تلخيص النصوص تلقائيًا باللغة العربية.

الكلمات المفتاحية:

تلخيص النصوص الآلي، تلخيص تجريدي، اللغة العربية،mBART ، المحولات، معالجة اللغة الطبيعية، التعلم العميق، XLSUM.

List of Abbreviations:

AI: Artificial Intelligence

ANN: Artificial Neural Networks

API: Application Programming Interface

ATS: Automatic Text Summarization

BART: Bidirectional and Auto-Regressive Transformers

BERT: Bidirectional Encoder Representations from Transformers

Bi-LSTM: Bidirectional Long Short-Term Memory

BLEU: Bilingual Evaluation Understudy

BRNN: Bidirectional Recurrent Neural Network

CNN: Convolutional Neural Networks

CPU: Central Processing Unit

DNN: Deep Neural Networks

DL: Deep Learning

EASC: Essex Arabic Summaries Corpus

ESA: Explicit Semantic Analysis

FFNN: Feed-Forward Neural Network

GP: Genetic Programming

GPU: Graphics Processing Unit

GPT: Generative Pre-trained Transformer

GRU: Gated Recurrent Unit

LSA: Latent Semantic Analysis

LSTM: Long Short-Term Memory

MDS: Multi-Document Summarization

ML: Machine Learning

MLM: Masked Language Modeling

MSA: Modern Standard Arabic

MT: Machine Translation

NLP: Natural Language Processing

NLG: Natural Language Generation

NLU: Natural Language Understanding

NSP: Next Sentence Prediction

OOV: Out-of-Vocabulary words

QA: Question Answering

RNN: Recurrent Neural Network

ROUGE: Recall-Oriented Understudy for Gisting Evaluation

RST: Rhetorical Structure Theory

RST-trees: Rhetorical Structure Trees

SA: Sentiment Analysis

SDS: Single-Document Summarization

seq2seq: Sequence To Sequence

SRL: Semantic Role Labeling

SVD: Singular Value Decomposition

T5: Text-To-Text Transfer Transformer

TF-IDF: Term Frequency-Inverse Document Frequency

TPU: Tensor Processing Unit

VSM: Vector Space Model

Contents

General introduction……2
Problematic and Motivation ……3
Objective ……3
CHAPTERI: STATE OF ART ……
1.1 Introduction ……5
1.2 Artificial Intelligence ……5
1.3 NLP ……6
1.3.1 Problem definition ……6
1.3.2 Applications of NLP ……6
1.4 Text Summarization ……7
1.4.1 What is Text Summarization ……8
1.4.2 Manual text summarization ……8
1.4.3 Automatic text summarization ……8
1.4.4 Manual vs Automatic Text Summarization ……9
1.5 Arabic Language and its Challenges ……10
1.5.1 Arabic Language …… 10
1.5.2 Arabic Language challenges …… 11
1.6 Automatic Text Summarization ……14
1.6.1 Categorization of Automatic Text Summarization ……14
1.6.2 Automatic text summarization approaches ……17
1.7 Evaluation of Automatic Text summarization system ……24
1.7.1 Evaluation techniques ……24
1.8 RELATED WORKS ……25
1.8.1 Abstractive Arabic Text Summarization ……26
1.8.2 Extractive Arabic Text Summarization ……27
1.9 Conclusion ……30
CHAPTER II: Deep Learning and Text Summarization……
2.1 Introduction ……31

2.2 Artificial Neural Network (ANN) ……31
2.3 Deep learning ……33
2.4 Deep Neural Networks (DNN) ……33
2.5 Deep Neural Networks (DNN) architectures ……33
2.5.1 Convolutional neural network CNN ……33
2.5.2 Recurrent neural networks (RNN) ……34
2.5.3 Bidirectional recurrent neural networks (BRNN) ……35
2.5.4 Long Short-Term Memory (LSTM) ……36
2.5.5 Gated recurrent units (GRUs) ……36
2.6 Encoder-Decoder ……37
2.7 Sequence-to-sequence models ……38
2.8 Attention mechanism ……39
2.9 Transformers ……39
2.10 Pre-trained models Based on Transformers ……40
2.11 Conclusion ……47
CHAPTER Ⅲ: MODEL CONCEPTION……
3.1 Introduction ……49
3.2 Methodology ……49
3.3.1 Dataset ……50
3.3.2 Tokenization ……53
3.3.3 Fin-tuning ……54
3.3.4 Test ……54
3.3 Proposed Approach……55
3.4 Model ……58
3.5 Conclusion ……62
CHAPTER Ⅳ: IMPLEMENTATION, RESULTS AND BENCHMARKING……
4.1 Introduction ……64
4.2 Implementation environment and tools ……64
4.2.1 Google Colaboratory ……64
4.2.2 Python ……65
4.2.3 PyTorch ……65
4.3 Implementation phase ……66

4.3.1 Loading and preprocessing the dataset ……..66
4.3.2 Fine-tuning the model ……69
4.3.3 Prediction ……74
4.4 Evaluation of model ……74
4.5 Development interface ……76
4.6 Conclusion ……78
General Conclusion and future work ……..79
Bibliography……

List of Figure:

Figure 1.1: Manual text summarization 8
Figure 1.2: Automatic Text Summarization 9
Figure 1.3: Letter forms in Arabic language. 11
Figure 1.4: letter Marks in Arabic language 11
Figure 1.5: Arabic digits 12
Figure 1.6: Example of Arabic inflection 14
Figure 1.7: Classification of Automatic Text Summarization Systems 14
Figure 1.8: The architecture of Arabic text summarization systems. 22
Figure 2.1: Model of an artificial neuron 33
Figure 2.2: Neural network architecture 34
Figure 2.3: Architecture of a typical convolutional neural network 35
Figure 2.4: A simple recurrent neural network (RNN) 36
Figure 2.5: A bidirectional sequence learning model with one hidden layer in the unfolded form. 36
Figure 2.6: LSTM based RNN architectures with a recurrent projection layer and an optional non-recurrent projection layer 37
Figure 2.7: GRU unit cell (repeating module in a GRU network). 38
Figure 2.8: Architecture of Encoder Decoder 39
Figure 2.9: Sequence to sequence architecture 39
Figure 2.10: The basic seq2seq model. SOS and EOS represent the start and end of a sequence, respectively 40
Figure 2.11: Overview of Transformer architecture 41
Figure 2.12: Feeding sentences with missing words to the BERT model for training. 42
Figure 2.13: Overall pre-training and fine-tuning procedures for BERT. [61] 43
Figure 2.14A: BERT. 45
Figure 2.14B: GPT 45
Figure 2.14 C: BART 46
Figure 2.15: Transformations for noising the input that we experiment with 46
Figure 2.16: multi-tasking concept of the T5 model 47
Figure 3.1: the methodology 52

Figure 3.2: example of tokenization 56
Figure3.3: BART encoder decoder architecture 60
Figure 3.4: Our model architecture 62
Figure 4.1: Version of python used in this work 65
Figure 4.2: load the dataset 66
Figure 4.3: Selecting Relevant Columns and Splitting. 67
Figure 4.4: the functions used to remove null and duplicate rows. 67
Figure 4.5: the function used to remove diacritics. 67
Figure 4.6: the function used to normalize Arabic letters. 68
Figure 4.7: The functions of text cleaning. 68
Figure 4.8: Load the associated pre-trained tokenizer of the model 69
Figure 4.9: the preprocess function used 69
Figure 4.10: Load the chosen model 70
Figure 4.11: Defining training arguments 71
Figure 4.12: Defining the data collator 71
Figure 4.13: the compute_metrics function 73
Figure 4.14: Defining the Trainer and strat training 74
Figure 4.15: Measurements of 3 epochs 74
Figure 4.16: Apply the prediction function on the test dataset 74
Figure 4.17: Evaluation of our model using ROUGE metric 75
Figure 4.18: The main page of our interface 76
Figure 4.19: The automatic summarization page of our model 76
Figure 4.20: Application of our model on text from the web using our interface 77

List of Tables:

Table 1.1: Manual vs Automatic Text Summarization [11]..... 9
Table 1.2: A sample of letters with their different letters shapes. [13] 12
Table 2.1: Different version of AraBERT……...…….43
Table 1.3: The eight main Arabic diacritics. [16] 13
Table 4.1: The Seq2SeqTrainingArguments used..... 70
Table 4.2: A comparison of our model with previous works..... 75

General introduction:

With the current availability of information and texts in a wide range, we are surrounded by large amounts of textual data not in a metaphoric sense; it is a physical reality, such as newspapers, scientific papers, books, journals, documents, social media posts, and more.

Textual information in electronic form accumulates quickly and in very large quantities , In this vast ocean of digital information that are left unusable unless we find a way to make it available for users , it becomes clear that effective summarization mechanisms are indispensable for distilling essential content from an enormous sea of words.

Text summarization is the task of creating a short and concise summary of a longer text, such as an article or a document, which preserves the main ideas and information of the original text. Text summarization is an important and useful application of natural language processing (NLP), as it can help users to quickly access and understand large amounts of textual data.

Arabic data is a subset of the data available on the internet that is written in Arabic, which is one of the most widely spoken languages in the world, Arabic data is diverse and rich, as it covers various fields and disciplines, such as culture, religion, history, politics, science, and technology. Arabic data is also valuable and influential, as it can shape the opinions and behaviors of millions of people across the Middle East and North Africa, and beyond. However, Arabic data is also underrepresented and underutilized, as it accounts for only a small fraction of the data on the internet, and faces many challenges and barriers in terms of access, quality, and analysis.

Arabic text summarization is a specific case of text summarization that deals with texts written in Arabic. Arabic text summarization is a relevant and interesting problem to solve, as it can benefit various domains and applications, such as education, journalism, business, health, and entertainment. However, Arabic text summarization also faces some challenges and difficulties, such as the complexity and richness of the Arabic language, the lack of standardized and annotated corpora, the scarcity of linguistic resources and tools, and the diversity and variation of the Arabic dialects.

Problematic and Motivation:

Arabic text summarization faces some challenges and difficulties, such as the complexity and richness of the Arabic language, the lack of standardized and annotated corpora, the scarcity of linguistic resources and tools, and the diversity and variation of the Arabic dialects. To address these challenges, various approaches and methods have been proposed and applied, such as statistical, semantic, linguistic, and deep learning methods. These methods use different features and techniques to measure the importance and relevance of the text units, and to generate coherent and informative summaries.

However, despite the advances and achievements of deep learning for Arabic text summarization, there remains scope for further improvement and innovation. The existing deep learning models for Arabic text summarization suffer from some limitations and drawbacks, such as the low quality and diversity of the generated summaries, the lack of coherence and cohesion of the summary text, the difficulty of handling out-of-vocabulary words and rare words, the high computational cost and complexity of the models, and the vulnerability of the models to noise and errors.

Therefore, the main research problem and question of this project is: How can we design and develop a novel and effective deep learning model for Arabic text summarization that can overcome the limitations and challenges of the current approaches and generate high quality, coherent, and diverse summaries of Arabic texts?

Objective:

The main objective and contribution of this project is to propose and implement a new deep learning model for Arabic text summarization that can address the research problem and question. The proposed model will be based on the state-of-the-art deep learning techniques and architectures, such as BERT, and will incorporate some novel and innovative features and mechanisms, such as semantic similarity, lexical diversity, discourse structure, and reinforcement learning. The proposed model will be evaluated and compared with the existing deep learning models for Arabic text summarization, using various metrics and criteria, such as ROUGE and BLEU. The proposed model is expected to outperform the existing models and generate summaries that are more accurate, fluent, coherent, and diverse than the current state-of-the-art.

CHAPTER I:

STATE OF ART

1.1 Introduction:

Over the past twenty years, there has been a significant increase in the demand for automated document summarization. This surge is primarily fueled by the abundance of online textual content, including news articles and academic papers. Entrusting this task to machines has become essential to efficiently sift through vast amounts of text and extract crucial information. Although text summarization has received considerable attention for widely spoken languages such as English, the intricacies of Arabic, both grammatically and morphologically, have resulted in its neglect within this research domain.

Assessing the quality of machine-generated summaries is vital for determining their reliability. Traditionally, this evaluation has relied on human judgment, which is impractical for large datasets. Alternatively, various metrics offer an automated approach.

In this section, we establish the foundation for our study by defining the Artificial Intelligence (AI) and then delving into its subfield, Natural Language Processing (NLP) and examining its practical applications. We explore both manual and automated summarization processes, categorize Automatic Text Summarization Systems and automatic text summarization approaches, and discuss the unique attributes of the Arabic language, its encoding, and the challenges it presents for Natural Language Processing (NLP) applications. Finally, we review recent progress in text summarization and introduce evaluation metrics.

1.2 Artificial Intelligence:

Artificial intelligence (AI) is defined as intelligence exhibited by an artificial entity to solve complex problems and such a system is generally assumed to be a computer or machine. Artificial Intelligence is an integration of computer science and physiology Intelligence in simple language is the computational part of the ability to achieve goals in the world. Intelligence is the ability to think to imagine creating memorizing and understanding, recognizing patterns, making choices adapting to change and learn from experience. Artificial intelligence concerned with making computers behave like humans. [1]

Since the advent of digital computers in the 1940s, it has been demonstrated that computer programs can be designed to solve complex tasks such as proving mathematical theorems and even playing chess. [2]

1.3 NLP:

1.3.1 Problem definition:

In a world where human language is the primary mode of communication, enabling machines to understand and process this language has become a fundamental challenge. Natural Language Processing (NLP) emerges as the solution to this challenge, a field dedicated to equipping computers with the ability to intelligently comprehend and generate human language. By bridging the gap between human communication and computational logic, NLP opens the door to a wide array of practical applications and transformative technologies.

Natural Language Processing (NLP) constitutes a pivotal domain within Artificial Intelligence (AI), focusing on endowing computer systems with the capability to analyze, comprehend, and decipher human language in both spoken and written forms. By leveraging textual or vocal inputs, NLP delves into multiple facets of linguistic analysis, encompassing syntactic structures (the contextual organization of words and phrases) as well as semantic nuances (the underlying meanings). It employs an amalgamation of methodologies including rule-based approaches, statistical inference, machine learning (ML), and deep learning (DL) models to facilitate the processing of human language, thereby furnishing pertinent outputs. [3]

Fundamentally, NLP comprises two primary components: Natural Language Understanding (NLU) and Natural Language Generation (NLG). NLU delves into various linguistic disciplines, including Phonology for sound analysis, Morphology for studying word formation, Syntax for understanding sentence structure, Semantics for grasping linguistic meanings, and Pragmatics for contextual comprehension. On the other hand, NLG involves the creation of coherent phrases, sentences, and paragraphs based on internal representations, ensuring the generation of meaningful linguistic outputs. [4]

1.3.2 Applications of NLP:

Over time, NLP has witnessed widespread adoption across diverse domains, fundamentally transforming our interactions with technology and data. Below are some of the notable applications of NLP:

- **Machine Translation (MT):**

It utilizes natural language processing to autonomously translate between different natural languages through machine-driven processes. The complexity of machine translation

lies not merely in word-for-word conversion, but rather in preserving the genuine meaning of sentences, including grammar and tense structures. [4]

- **Chatbots:**

A Chatbot refers to a communication tool designed to mimic human conversation through either auditory or textual interfaces. This technology is alternatively termed a conversational agent, an intelligent agent, or a dialogue system. [5]

- **Question Answering (QA) :**

It is one of the most important natural language processing (NLP) tasks. Its objective lies in leveraging NLP techniques to produce an appropriate response to a provided query, drawing from extensive unstructured textual data. [6]

- **Sentiment analysis (SA) :**

This process falls under the realm of natural language processing and entails monitoring public sentiment regarding a specific product or subject matter. Known as sentiment analysis or opinion mining, it encompasses constructing a system to gather and analyze opinions expressed in various mediums such as blog posts, comments, reviews, or tweets. [7]

- **Summarization:**

Summarization is an expansive and varied domain. Traditionally, it is regarded as a subset of natural language processing, but many innovations trace their roots back to information retrieval techniques.

Summarization systems generate condensed outputs that convey important information contained in one or more sources for particular users and tasks. In principle, input sources and system outputs are not limited to text, but this entry focuses exclusively on generating textual summaries from textual sources. [8]

Summarization systems generate succinct outputs that capture key information from singular or multiple sources, customized to meet specific user needs and goals. While input sources and system outputs could theoretically encompass various formats beyond text, this discourse focuses solely on generating textual summaries from textual sources.

1.4 Text Summarization:

As the digitalization of text continues to expand, particularly with the growth of the Internet, individuals are facing an inundation of data. Effectively and efficiently extracting valuable information from this vast amount of data has become increasingly crucial. Various

techniques have been introduced to address this challenge, and text summarization is one of them.

1.4.1 What is Text Summarization?

Text summarization is the process of shortening text computationally, to create a subset that represents the most significant information within the original content. Text summarization tries to find the most informative sentences in an article and extract this information [9]. There are two types of Text summarization: Manual text summarization and Automatic text summarization.

Text summary can be classified into either single-document or multi-document summarization based on the number of documents to be summarized. In multi-document systems the documents must have a topic-relation with each other, and then the summarizer generates a summary based on these documents. In single-document systems the summarization is built using that single-document only [10].

1.4.2 Manual text summarization:

Manual text summarization is the process of condensing the content of a longer text into a shorter version through human effort, typically involving rewriting or rephrasing to capture the essential information. It is illustrated in **Figure 1.1**:

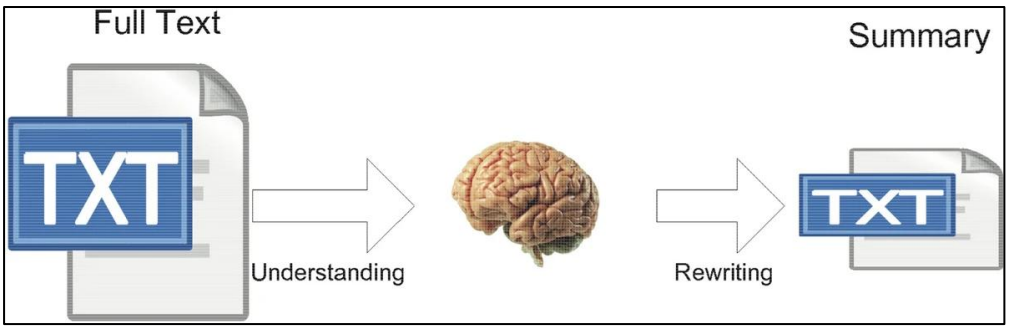


Figure 1.1: Manual text summarization [11]

Initially, a complete text is provided as input. A person reads and comprehends the entire text, then rewrites it into a shorter form. The resulting summary, created through manual text summarization, contains several sentences that may not have been explicitly present in the original text. [11]

1.4.3 Automatic text summarization:

Automatic Text Summarization (ATS) is a process of automatically extracting a shorter version of the original text with preserving its information content and overall

meaning [10] , it has been applied to various domains in order to find the most relevant information and make critical decisions quickly. It is illustrated in **Figure 1.2**:

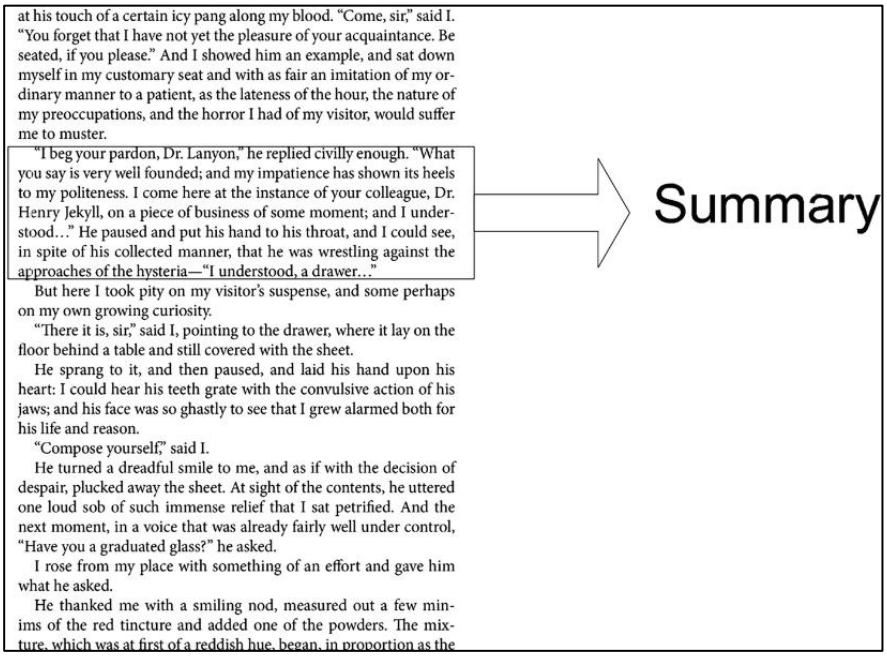


Figure 1.2: Automatic Text Summarization [11]

The input for automatic text summarization is the complete text, similar to manual summarization. The text is divided into sentences or paragraphs using punctuation marks or line breaks. Certain paragraphs or sentences are then chosen as the key components. Preferring paragraph selection over individual sentences is often favored for creating the summary. [11]

1.4.4 Manual vs Automatic Text Summarization:

The comparisons of the two kinds of text summarizations are illustrated in **Table 1.1:**

	Manual	**Automatic**
Definition	Rewriting briefly	Paragraph or sentence selection
Preliminary task	Understanding	Partition
Mass summarization	Difficult	Possible
Writing quality	Good	Plain

Table 1.1: Manual vs Automatic Text Summarization [11]

Manual summarization involves comprehending and condensing the full text, while automatic summarization entails selecting key paragraphs or sentences. In manual summarization, understanding the entire text is essential, whereas in automatic

summarization, segmenting the text is crucial. Automatic text summarization is useful for summarizing numerous texts individually, but it may not consistently produce high-quality summaries. [11]

1.5 Arabic Language and its Challenges:

1.5.1 Arabic Language:

Before delving into the intricacies of the Arabic language, let's start by introducing it.

Arabic is the sixth most spoken language globally. It's closely linked with Islam, with over 200 million Muslims praying in Arabic five times a day. It's also the primary language in Arab countries, carrying global importance. Arabic is linguistically unique, belonging to the Semitic family, unlike the Indo-European languages spoken in the West [12], Therefore, our study focused on using ATS for Arabic text documents to enrich the research on NLP for Arabic text summarization.

There are three types characterize Arabic: Classical (Traditional or Quranic) Arabic, Modern Standard Arabic and Dialect Arabic, These variations are distinguished by differences in morphology, syntax, and vocabulary. [12]

- **Classical Arabic:** Classical Arabic is mainly used in Arabic-speaking nations rather than among Diaspora communities. It appears in religious texts like the Sunnah and Hadith, as well as many historical documents. Diacritic marks, also known as "Tashkil" or short vowels, are frequently employed in Classical Arabic to aid pronunciation. In contrast, diacritics are often optional in other forms of Arabic writing**.** [12]
- **Modern Standard Arabic:** Modern Standard Arabic (MSA) finds its place in television, newspapers, poetry, and literature. Courses at institutions like the Arab Academy often teach Arabic in this standardized form. MSA has the flexibility to incorporate new terms, especially related to science and technology. However, the written Arabic script has remained unchanged in terms of alphabet, spelling, and vocabulary for at least four millennia, making it unique among living languages. [12]
- **Dialect Arabic:** Colloquial Arabic, also known as Dialect Arabic, is commonly spoken by Arabs in their daily lives across different countries and regions. It encompasses variations such as Mesopotamian, Arabian Peninsula, Syro-Palestinian, Egyptian, and Maghrebi Arabic. This form of Arabic is often used in informal communication, including on the

internet and social media platforms. While it differs from region to region, colloquial Arabic sometimes borrows elements from Modern Standard Arabic (MSA). [12]

The Arabic script employs two main types of symbols for writing words: letters and diacritics. Additionally, it includes supplementary symbols such as digits and punctuation marks.

There are 19 distinct letter shapes as shown in **Figure 1.3**:

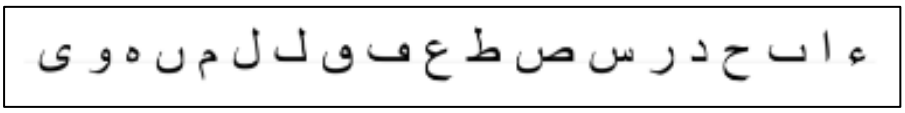
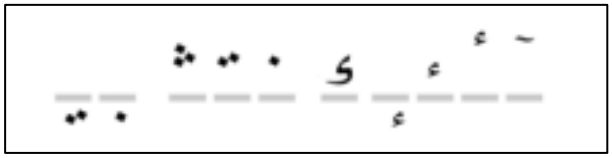


Figure 1.3: Letter forms in Arabic language. [13]

The letters Marks also called consonantal diacritics can be sub-classified into three



types: dots, short kaf and Hamza as shown in **Figure 1.4**:

Figure 1.4: letter Marks in Arabic language [13]

There are 36 Arabic letters used in MSA can be classified into the following subsets:

- **The basic 28 letters:** The fundamental Arabic alphabet consists of 28 letters, each representing one of the 28 consonantal sounds in Arabic. With the exception of the Hamza letter form, these letters encompass all letter forms. The markings on each letter are highly discriminative, enabling differentiation between distinct sounds.
- **The Hamza letters:** Six types of Hamza exist, with the first being the "Hamza-on-the-Line" formed by the Hamza letter itself (ء). The remaining five utilize various letter forms combined with the Hamza and Madda letter marks (أ إ آ ؤ ئ) .
- **The Ta-Marbuta:** This letter serves as a distinct morphological marker indicating a feminine ending. The Ta-Marbuta (ة) is a composite letter merging the forms of Ha (ه) and Ta (ت).

The Ta-Marbuta exclusively appears at the end of words. Meanwhile, the letter Ta (ت) is utilized when the morpheme it signifies is in a word-medial position.

- **The Alif-Maqsura:** The Alif-Maqsura (ى) is another distinctive morphological identifier, signifying various aspects from feminine endings to underlying word roots. It's a composite letter blending the forms of Alif (ا) and Ya (ي). The Alif-Maqsura emerges solely

as a dotless Ya at the end of words. However, when the morpheme it denotes is in a word-medial position, either Alif or Ya is employed. [13]

1.5.2 Arabic Language challenges:

When realizing an Arabic NLP project we should know that there are many challenges that we must take into consideration. In what follows, we will cite the most important among them.

– Arabic language is a Semitic language written and read in a cursive script from right to left. While text is written from right to left, numbers can be written and read from right to left or from left to right. [14]

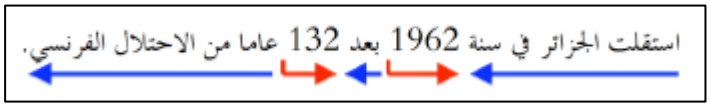
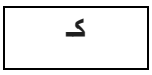
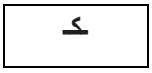


Figure 1.5: Arabic digits [13]

– Arabic is composed of 28 letters that change their shapes according to their position [14], For example, the letter (ك), transliterated as *kãf*, if it appears at the beginning of a word it can be written as:



If it appears in the middle:



If it appears in the end:



The **table 1.2** illustrates the different letters shapes according to their position:

`	*w*	*r*	*d*	*A*	*l*	*k*	*h*	*T*	*S*	*s*	*q*	*f*	*m*	*γ*	*j*	*y*	*n*	*b*	
ء	و	ر	د	ا	ل	ك	ه	ط	ص	س	ق	ف	م	غ	ج	ي	ن	ب	Isolated
					لـ	كـ	هـ	طـ	صـ	سـ	قـ	فـ	مـ	غـ	جـ	يـ	نـ	بـ	Initial
	ـو	ـر	ـد	ـا	ـل	ـك	ـه	ـط	ـص	ـس	ـق	ـف	ـم	ـغ	ـج	ـي	ـن	ـب	Medial
					ـل	ـك	ـه	ـط	ـص	ـس	ـق	ـف	ـم	ـغ	ـج	ـي	ـن	ـب	Final

Table 1.2: A sample of letters with their different letters shapes. [13]

- Arabic has no capital letters, which makes it difficult to extract nouns for annotation in the summarization process. [14]
- In English and other languages that use the Latin script, sentences typically starts with a capital letter and end with a period. This structure is leveraged in various natural language processing (NLP) tasks like machine translation, information retrieval, clustering, and summarization, where accurate sentence segmentation is crucial. However, languages such as Arabic, Chinese, Japanese, and Korean lack capitalization and have less strict punctuation rules. This absence of clear markers makes preprocessing text in these languages significantly more challenging. [15]
- Arabic uses diacritical marks to clarify the meaning of a word in a sentence because the same word can have several meanings; for example, the word (عقد) can mean complicated, contract, or necklace. Unfortunately, most Arabic texts do not use diacritical marks, leaving the reader to deduce the meaning of a word through its context. This can cause words to be annotated incorrectly during the summarization process [14], the table 1.3 illustrate the main Arabic diacritics:

Fatha	ـَ	Kasrah	ـِ
Dhammah	ـُ	Sukkon	ـْ
Shaddah	ـّ	Tanween Fath	ـً
Tanween Kasr	ـٍ	Tanween Dham	ـٌ

Table 1.3: The eight main Arabic diacritics. [16]

- Words in Arabic have roots and affixes, and although the latter can help to clarify meaning, they can also change words' meanings entirely. For example, the root (درس) means study; with the addition of the letter (م), it becomes (مدرس), which means teacher. This limits the overall number of root vocabularies that summarization models can use to generate summaries, as a word can be repeated many times with different affixes.
- Arabic's complex morphology makes it possible to convey a whole sentence with only one word; for example, the word (أنلزمكموها) means 'shall we compel you to accept it' (**Figure 1.6**). Such words make ATS models extremely challenging to develop. [14]

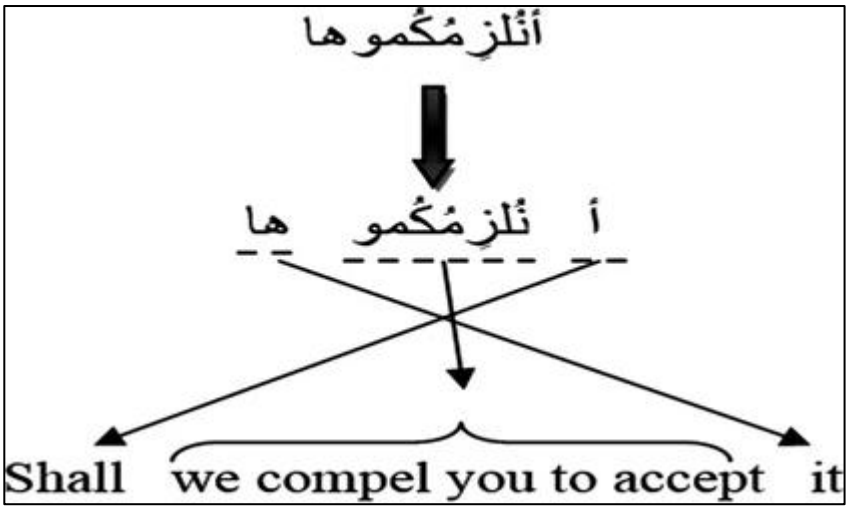


Figure 1.6: Example of Arabic inflection. [17]

1.6 Automatic Text Summarization:

1.6.1 Categorization of Automatic Text Summarization:

An automatic text summarization system can be classified using many different factors and criteria as shown in **Figure 1.7**:

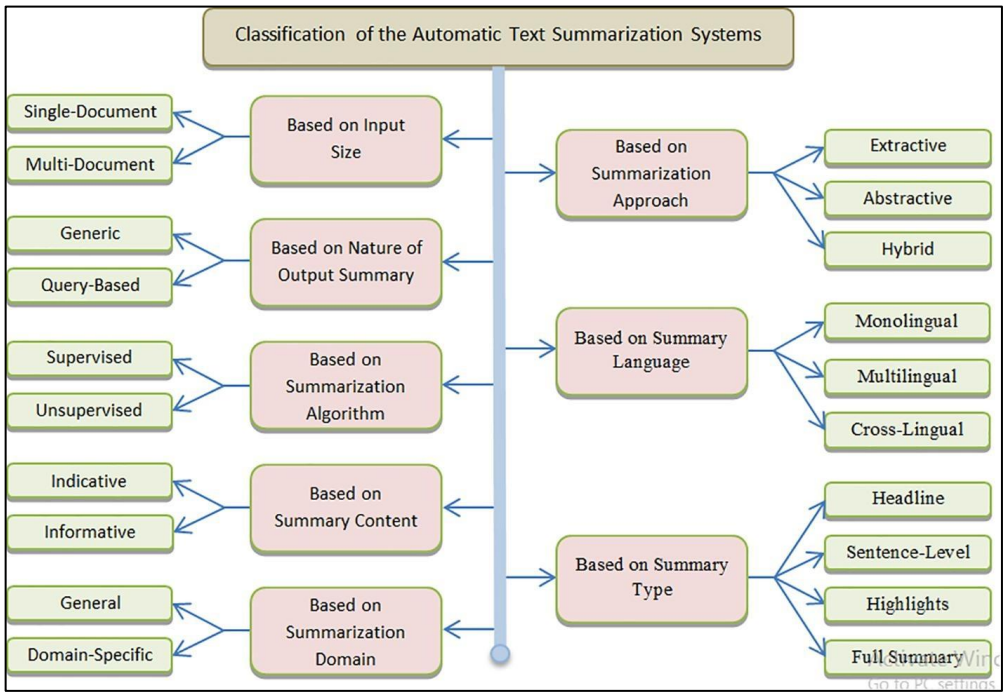


Figure 1.7: Classification of Automatic Text Summarization Systems. [18]

A. Based on Input Size:

Based on input size there are two types of text summarization single document summarization and multi-document summarization:

- **Single document:** Single-document text summarization refers to the automated condensation of information contained within a single document. [19].Single-Document Summarization (SDS) involves utilizing a single text document to produce a summary, aiming to condense the input document while retaining crucial information. [18]
- **Multi-document:** Multi-document text summarization refers to the automated condensation of information extracted from multiple documents [19].In Multi-Document Summarization (MDS), the summary is created by analyzing a collection of input documents, with the aim of eliminating redundant content present across the documents. [18]

Single-document text summarization finds application in specific contexts such as generating appropriate titles or summaries for given passages, as in comprehension tasks. On the other hand, multi-document text summarization is often employed in broader scenarios like aggregating news articles from various sources for news summarization purposes. [20]

B. Based on Output Summary Nature:

- **Generic summary:** A generic text summarizer is designed to extract key information from one or more input documents, offering a broad understanding of their content. A generic summary provides a comprehensive overview of the document content, capturing its essence. [18]. In generic summarization, the system generates summaries without any specific user query or context.
- **Query-based summary:** Query-based summarization involves a multi-document summarizer processing a collection of similar documents, which are retrieved from a larger corpus in response to a query [18]. Consequently, the produced summary incorporates content relevant to the query. A query-based summary prioritizes information that directly pertains to the initial search query, ensuring relevance to the user's information needs. [18].Query-based summarization involves generating summaries in response to a user query or specific input.

C. Based on content:

Automatic text summarization is classified as informative and indicative summary.

- **Informative:** An informative summary encapsulates the essential information and key ideas from the original text, ensuring comprehensive coverage of all

topics addressed in the text [18]. These summaries effectively communicate the critical information present in the source document [20].

- **Indicative:** An indicative summary provides a brief overview or general information about the source text, serving to alert the user to the content of the input text [18]. These summaries offer a descriptive glimpse into the document's subject matter [20].

D. Based on approaches:

Summarization methods can be categorized into three types based on how summaries are produced [19]:

- **Extractive:** Extractive text summarization involves directly selecting and concatenating sentences from the given corpus to form a summary.
- **Abstractive:** Abstractive text summarization entails paraphrasing the content of the given corpus and generating new sentences to form a summary.
- **Hybrid:** Hybrid summarization combines both extractive and abstractive methods. This approach involves extracting some sentences directly from the corpus while also generating new sentences to compose a summary.

E. Based on language:

The ATS system can be classified into three types based on the language used in input and output [20]:

- **Monolingual:** In this type, both the input document and the output summary are in the same language.
- **Multilingual:** Here, the input may consist of various languages, and the summary is generated in the corresponding languages.
- **Cross lingual:** In this category, the input document is in one language (such as English), while the output summary is in a different language.

F. Based on the Summary Type:

Based on the type of summary produced, text summarization can be categorized into four types: Headline, Sentence-Level, Highlights, or Full Summary. [18]

- **Headline summary:** This type generates a concise headline, typically shorter than a full sentence.
- **Sentence-level summary:** Sentence-level summarization produces a single sentence summarizing the input text, often using abstractive techniques.

- **Highlight summary:** Highlight summarization creates a highly condensed summary, often in bullet point format, providing a brief overview of the main information in the input document(s).
- **Full summary:** Full summary generation is guided by either the required summary length or a compression ratio, aiming to provide a comprehensive overview of the text.

G. Based on the Summarization Algorithm:

- **Supervised:** In order to select important content from documents in a supervised system, training data is required. This training data typically consists of a substantial amount of labeled or annotated data, which serves as input for learning techniques. In these systems, sentences are often treated as a two-class classification problem at the sentence level. Positive samples represent sentences belonging to the summary, while negative samples represent sentences that do not belong to the summary. [19]
- **Unsupervised:** Unsupervised systems operate without the need for training data. They generate summaries solely by analyzing the documents they aim to summarize. Consequently, they can be applied to new data without additional adjustments. These systems utilize heuristic techniques to extract pertinent sentences and form a summary. Clustering is a common method employed in unsupervised systems to group similar sentences or documents together. [19]

H. Based on the summarization domain:

Based on the summarization domain ATS can be classified as General or Domain-Specific summarization:

- **General:** The general or domain-independent ATS system summarizes documents that belong to different domains [18]
- **Domain-Specific:** In contrast, domain-specific ATS systems are tailored to summarize documents within a particular domain, such as medical or legal documents.

1.6.2 Automatic text summarization approaches:

The ATS system employs three primary approaches: Abstractive, extractive, and hybrid text summarization. These approaches are further subdivided into various subcategories [21]:

1.6.2.1 Abstractive Text Summarization Approaches:

Abstractive Text Summarization involves generating a concise overview of a source text, encapsulating its key points. The resulting summaries may incorporate novel phrases and sentences not present in the source material. Essentially, an abstractive summarization system generates fresh text that conveys the essence of the original while enhancing coherence, readability, and minimizing redundancy. Consequently, this task presents challenges in producing summaries that emulate or closely resemble those crafted by humans. [21]

In this section, three categories of abstractive automatic text summarization methods are outlined. The initial category relies on structural elements such as graphs, trees, rules, and compilations. The second category is rooted in semantics, employing techniques like semantic text representation and common language frameworks. The third category employs deep learning as its primary strategy. [22]

A. Graph-Based Approach:

The graph-based method involves several preparatory steps, including creating a graph, constructing a textual graph to depict the original text, and generating the summary. This method is applicable across various domains and does not require human expertise. By linking all words along a path on the word graph, a new phrase can be generated. However, a drawback of this approach is that word graphs may not accurately represent the meanings of the words. Additionally, because of the way nodes are stored, it is challenging to combine sentences composed of nodes. [22]

In the graph-based approach, the document is represented in the form of undirected graph. For every sentence, there is a node. An edge between two nodes is drawn if there is a relation between these two nodes. A relation can be a cosine similarity above a threshold, sharing a common word, or any other type of relationships. After drawing a graph, it is possible to view the sub-graphs of connected nodes as a cluster of distinct topics covered in the document. [17]

B. Tree-Based Approach:

These algorithms identify comparable statements and merge them to generate an abstractive summary. Similar sentences are depicted through a tree structure, with dependency trees being the most commonly utilized representation for text. Trees undergo management processes such as pruning and linearization (conversion to strings), among others. The method

offers advantages such as producing summaries of enhanced quality as language generators yield fewer redundant and more fluent summaries. However, it requires identifying common terms to discern relationships between sentences, which may overlook crucial contextual phrases. Additionally, its effectiveness is constrained by the quality of available parsers, and it tends to prioritize syntax over semantics. [22]

C. Rule-Based Approach:

These methodologies require the establishment of rules and categories to identify the most crucial concepts within the input text. The stages of this method are as follows: Initially, categorize the input text by words and ideas to construct an abstractive summary. Subsequently, formulate questions based on the domain of the input text, and then search the text for terms and ideas to respond to these questions. The resulting summaries are rich in information. Handling additional data involves increasing the complexity and diversity of the abstraction scheme. [22]

D. Ontology-Based Approach:

In every domain, there exists a distinct array of articles, each structured with its own information schema. The fundamental concept involves leveraging an ontology to extract pertinent information from a text and generate an abstract summary. This approach relies on publications within a specific domain and is capable of handling textual uncertainty while furnishing logical summaries. However, it necessitates the construction of a domain-specific ontology, a process that consumes time, thereby extending the overall time requirement. [22]

E. Semantic-Based Approach:

One widely used extractive automatic text summarization (ATS) technique based on semantics is Latent Semantic Analysis (LSA). LSA represents text semantics through the observed co-occurrence of terms, employing an unsupervised method. In the sentence scoring process of an LSA-based extractive summarizer, it initiates with the creation of the input matrix (term-to-sentence matrix) and then proceeds with Singular Value Decomposition (SVD) applied on the input matrix to establish associations between terms and sentences. Additional semantic-based ATS methods encompass Explicit Semantic Analysis (ESA) and Semantic Role Labeling (SRL).

F. Deep-Learning-Based Approaches:

Sequence-to-sequence learning (seq2seq) has enabled abstract summarization, proving successful in various NLP applications like machine translation, speech recognition, and

conversation systems. RNN models utilizing attention encoder-decoder mechanisms effectively summarize content. However, deep learning approaches encounter challenges including repetitive words or phrases and difficulties with out-of-vocabulary words (OOV), i.e., unusual or infrequent terms. Optimization of the cost function is achieved using the Adam optimizer (loss). Deep learning models are commonly applied for concise text summaries. To enhance abstractive summaries, it's advisable to integrate multiple approaches and strategies. Combining results from various ATS techniques holds promise for significantly improving summaries beyond what individual algorithms can achieve. These methods can be used in preprocessing to extract key terms from input text, which are then employed in generating the abstractive summary. [22]

1.6.2.2 Extractive Text Summarization Approaches:

A. Statistical-Based Methods:

These algorithms employ statistical analysis of specific features to identify the most crucial sentences and words within the original text. A common methodology involves calculating word frequency, which is widely utilized for scoring sentences. The score of a sentence is determined by its frequency count while disregarding stop words. A proposed strategy for generating news titles involves integrating word frequency, sentence position, and similarity metrics. In the realm of information retrieval, TF-IDF (term frequency-inverse document frequency) serves as a numerical measure to emphasize a term's significance within a document corpus. Summarization is achieved by considering features such as similarity to a centroid sentence, where the centroid sentence is derived using TF-IDF. Each sentence's similarity with the centroid is computed using cosine similarity, and the feature values of each sentence are then aggregated to derive the sentence scores. [22]

B. Structured-Based Approach:

This method primarily employs deep learning (DL) algorithms to identify the crucial elements within the original documents. This process may encompass various components such as the document's template, introductory phrases, body content, extraction rule framework, tree structures, and template-based structure. [21]

C. Semantic-Based Approach:

One prevalent method in semantic-based extractive ATS is latent semantic analysis (LSA), which utilizes observed term co-occurrences to represent text semantics in an unsupervised manner. The scoring process of each sentence in an LSA-based extractive summarizer starts with generating the input matrix (term-to-sentence matrix) and proceeds with applying Singular Value Decomposition (SVD) on this matrix to establish associations

between terms and sentences. Additional semantic-based ITS techniques encompass explicit semantic analysis (ESA) and semantic role labeling (SRL). [21]

D. Machine learning-based approaches:

In machine learning-based approaches, summarization strategies at the sentence level reframe the problem as a supervised classification task. Utilizing a trainingset comprising documents along with their corresponding human-generated summaries, the system learns to classify each sentence in a test document into either a "summary" or "non-summary" category. The sentence scoring process for the machine-learning-based summarizer involves: **a)** extracting features from the preprocessed document, incorporating various features of sentences and words, and **b)** inputting these extracted features into a neural network, which outputs a single value representing the score. [21]

E. Deep Learning Based approaches:

A deep learning-based summarization technique employing embedding focuses on document-level similarity, where each word's embedding captures its meaning. In this approach, a document is treated as a collection of sentences, and each sentence is treated as a collection of words. The task is formulated as maximizing a sub-modular function, defined by the negative sum of the distances between the closest neighbors in embedding distributions. Document-level similarity allows for more complex interpretations compared to sentence-level similarity. To capture key characteristics, a sentence-level selective encoding approach is employed, followed by the retrieval of summary sentences. [21]

F. Fuzzy logic-based approaches:

Fuzzy logic-based text summarization employs fuzzy logic principles to assess the relevance of phrases to the main theme of the text, offering a method for condensing lengthy texts. This technique evaluates each phrase's proximity to the primary topic, which is represented as a fuzzy set. Through fuzzy logic operations like fuzzy intersection, fuzzy union, and fuzzy complement, the algorithm determines the degree of relevance for each sentence. Subsequently, sentences with the highest relevance scores are chosen to compose the summary. This approach proves particularly useful when the primary theme is ambiguous or when the text includes unclear or imprecise terminology. It also shines in scenarios where phrase relevance varies depending on context or reader preferences. [21]

1.6.2.3 Hybrid Text Summarization Approaches:

Hybrid text summarization is a methodology that incorporates various approaches to create comprehensive summaries highlighting key points of a text. It combines extractive and abstractive techniques, where extractive summarization involves selecting crucial sentences or

phrases from the text, while abstractive summarization generates new text capturing the original meaning. Hybrid methods often integrate techniques from both approaches, leveraging the strengths of each. [21]

For instance, a hybrid approach may employ a neural network to generate an initial summary, refined further using extractive summarization techniques to ensure inclusion of essential information. While hybrid summarization offers more accurate and nuanced summaries compared to single-method approaches, it can be computationally intensive and complex. [21]

Advantages of hybrid text summarization include producing logical and insightful summaries by combining the strengths of both extractive and abstractive methods. However, challenges arise from complexity, resource requirements, and customization needs. The choice of the best hybrid technique depends on specific summarization goals and the trade-offs between content preservation and summary quality. Here are some merits and demerits of hybrid text summarization approaches. [21]

1.6.3 Architecture of automatic text summarization system:

Various types utilized in the design and implementation of automatic text summarization (ATS) systems. Summarized texts lack unit categorization, and summaries can be classified based on various criteria, as explored in numerous studies. The **Figure 1.8** illustrates the steps involved in extractive and abstractive text summarization using deep learning (DL) models: [22]

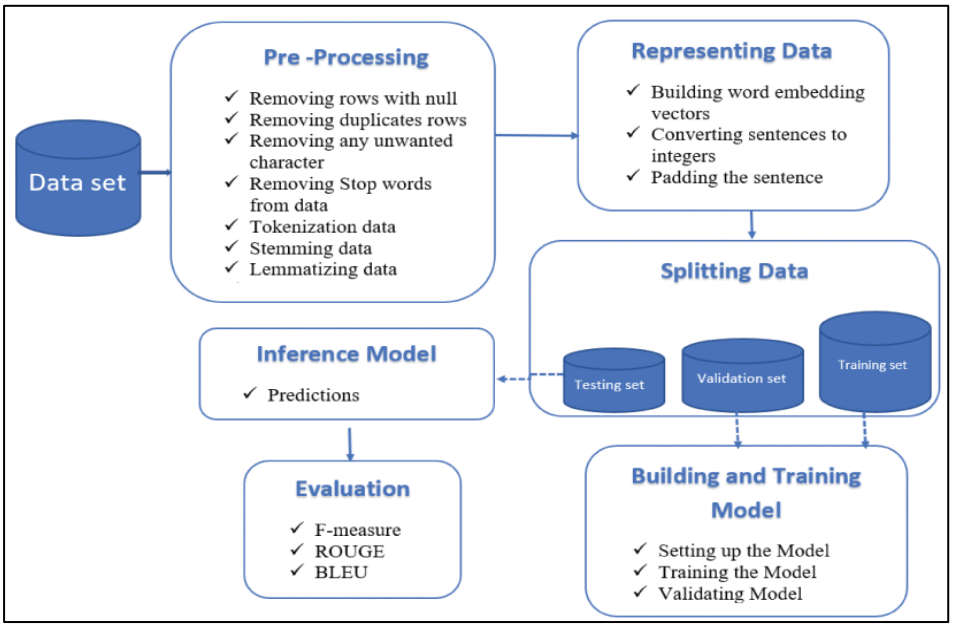


Figure 1.8: The architecture of Arabic text summarization systems. [22]

- **Preprocessing procedures:**

Preprocessing procedures entail identifying the specifications of the input text, including genre, language, multilingualism, subject specificity, summary size, and type. This step is necessary because raw texts often lack proper structure. Initially, rows with NULL values in either the new content or summary are eliminated, along with duplicate rows. Common techniques applied include segmentation, tokenization, stemming, stop words removal, elimination of unwanted characters, data normalization, and lemmatization. These steps aim to standardize and refine the data for subsequent analysis and summarization. [22]

- **Representing Data:**

To address the challenge posed by deep learning models that solely accept numerical inputs, word embedding is utilized. Word embeddings are constructed by concatenating each new content piece with its corresponding summary from a dataset of unique Arabic news. The dictionary now includes special tokens like INK, PAD, EOS, and SOS, which substitute less common or unknown words, pad short sentences, and mark sentence beginnings and endings. This approach enables faster model training by considering the lengths of both texts and summaries, thereby saving time and effort. Additionally, to ensure the model is trained using relevant data, any news articles containing more than one instance of INK or any summaries containing Dunks are excluded. Ultimately, each phrase in news articles and summaries is represented as an integer set. [22]

- **Splitting Data:**

During the splitting data phase, the dataset is partitioned into three subsets: training, validation, and testing. The training set is utilized to train the model, the validation set is employed for validation purposes, and the testing set, also known as the unseen set, is used for evaluating the model's performance. This division ensures that the model is trained, validated, and tested on distinct sets of data, preventing overfitting and providing reliable assessments of its performance. [22]

- **Building and Training Model:**

During the building and training phase, deep learning models are constructed by defining the architecture, which may include Seq2Seq, Bi-LSTM, RNN, BERT, among others. Learning parameters such as accuracy metric, loss function, optimizer, and the number of layers are specified. These layers enable the specification of transformations to be applied to the input sequence. [22]

Training the Model involves making predictions based on the current model state, calculating prediction errors, and updating network weights or parameters to minimize these errors and improve predictions. This process iterates until the model converges and can no longer improve. Training hyperparameters like learning rate, epochs, batch size, and early stopping are crucial in this process. [22]

Model validation is a critical step where the trained model is evaluated with a separate validation dataset, derived from the same dataset as the training set. The purpose of validation is to assess the model's generalization ability. Model validation occurs after training, aiming to identify the optimal model with the best performance. [22]

- **Inference Model:**

During the inference phase, the fully trained model undergoes testing to evaluate its performance. This process involves assessing the model's capabilities on a separate testing set. The testing set, comprising testing samples, is distinct from both the training and validation sets but follows the same probability distribution as the training set To test and evaluate all variations of the testing set, the testing data is inputted into the inference model to predict the summary. This step allows for the assessment of how well the trained model generalizes to unseen data and provides insights into its effectiveness in summarization tasks. [22]

1.7 Evaluation of Automatic Text summarization system :

Evaluation of the DL model involves assessing its quality using three standard metrics: F-measure, ROUGE, and BLUE. These metrics are non-differentiable and are well-suited for comparing the generated summary to the reference summary. They provide insights into the effectiveness and accuracy of the model's summarization performance. [22]

1.7.1 Evaluation techniques:

Evaluating text summarization presents challenges due to the complexity of discerning key phrases or essential content that enriches the summary. The placement of these key phrases can alter the summary's meaning depending on the contextual purpose, posing difficulties in identifying relevant information. Consequently, the development of automatic evaluation measures becomes imperative for ensuring reliable and efficient evaluation. Upon examining prior research on text summarization, various methods have been identified for evaluating summarization [23]. Subsequently, the evaluation metrics within the ATS domain are outlined below:

1.7.1.1 Extrinsic evaluation:

Extrinsic evaluation assesses the quality of the generated summary by examining its impact on other activities such as text categorization, information retrieval, and question answering. A good summary enhances these activities during the summarization process. Extrinsic evaluation encompasses various approaches. Relevance assessment gauges the text's pertinence to the topic, while reading comprehension evaluates its ability to answer multiple-choice assessments effectively. These evaluations provide insights into how well the summary aids in broader tasks beyond summarization. [23]

1.7.1.2 Intrinsic Evaluation:

Intrinsic evaluation assesses the quality of the summary by comparing machine-generated summaries with human-generated ones. Two primary factors, quality and information, are pivotal in judging a summary's effectiveness. Human experts may evaluate machine-generated summaries using various quality measures, including readability, non-redundancy, structure, coherence, and other metrics such as referential clarity, conciseness, focus, and content coverage. These assessments provide insights into the overall quality and informativeness of the machine-generated summaries. [23]

Some essential measures for intrinsic summary evaluation include precision, recall, and F-measure. Researchers must ensure comparability between human-generated and automatically generated summaries. Despite being equally good, it's possible for the two summaries to yield different evaluation outcomes using the mentioned metrics. The next section concentrates on the commonly used evaluation metrics in research, which include the following: [23]

- **Precision Metric:** The precision metric assesses the correctness of the percentage of sentences chosen by humans and the computer. It is calculated by dividing the total number of sentences between two summaries by the number of sentences in the system summary. [23]
- **Recall Metric:** The recall metric evaluates the system's ability to recognize how many sentences are selected by humans. It is calculated by dividing the number of sentences common between the reference and system summaries by the total number of sentences in the reference summary. These studies utilized recall metrics for evaluating the performance of the system. [23]
- **F-measure Metric:** The F-measure metric combines recall and precision metrics. It is calculated as the arithmetic mean of precision and recall. In the evaluation task, emphasis

is placed on the F-measure metric. [23], The basic way how to compute the F-score is to count a harmonic average of precision and recall [24]:

- **ROUGE Metric (Recall-Oriented Understudy for GISTING Evaluation):** The Recall-Oriented Understudy for Gisting Evaluation (ROUGE) metric is commonly used in automatic text summarization (ATS) and machine translation evaluations. It compares automatically generated summaries or translations with a set of predetermined summaries, such as human-generated summaries, ROUGE comprises five measures: ROUGE-N, ROUGE-L, ROUGE-W, ROUGE-S, and ROUGE-SU. [23]
- **BLEU (Bilingual Evaluation Understudy):** evaluation metrics are used to evaluate the quality of machine translation systems based on reference translations. The main task of this metric involves counting the number of n-gram matches found independently between the system's output and the reference translations. [23],

1.8 RELATED WORKS:

The text summarization is not recent; it dates back to the fifties (1950's) with Luhn's work [25], which laid the foundation for subsequent developments. Various methods have emerged, broadly categorized into statistical (involving features like TF/IDF, uppercase words, sentence length, similarity with the title, and sentence position), linguistic (utilizing frameworks like Rhetorical Structure Theory and Lexical chain), and Machine learning (employing techniques such as Neural Network and VSM). While these methods have yielded notable advancements in summarization, particularly for languages like English or French, research specific to Arabic remains scarce due to its intricate morphological and syntactic nature. This section provides an overview of existing works focusing on text summarization.
Most of the work that has been done for Arabic text summarization was based on the extractive method. [26]

1.8.1 Abstractive Arabic Text Summarization:

This section compares previous abstractive Arabic text summarization studies. In this study [27] Suleiman and Awajan introduced an attentional RNN-based sequence-to-sequence model featuring a two-layer bidirectional LSTM encoder and a one-layer LSTM decoder. Within this architecture, the initial layer of the encoder processes the word embeddings derived from the input text, while the subsequent layer handles the word embeddings of named entities. Their findings from the evaluation indicated that the inclusion of an additional layer in the encoder to handle named entities led to an improved representation of the input text, resulting in enhanced performance compared to a baseline model utilizing only one

layer. Additionally, they explored the utilization of two distinct pre-trained word embedding models and observed a significant impact of the embedding quality on the overall quality of the generated summaries.

In a recent study by Wazery et al. [28], various networks were evaluated for their effectiveness in Arabic abstractive summarization, including GRU, LSTM, and bidirectional LSTM with varying numbers of layers. Additionally, they examined the influence of text preprocessing techniques and the use of different word embedding models. Their findings revealed that employing a sequence-to-sequence model featuring a bidirectional LSTM encoder with three layers and a unidirectional LSTM decoder yielded superior performance compared to other model configurations tested in the study.

Sequence-to-sequence (seq2seq) learning has become the predominant architecture for abstractive text summarization, serving as the basis for many pioneering studies in the field, In 2015, Rush et al. [29]were among the first to utilize seq2seq for abstractive text summarization, introducing a version based on the attention mechanism. Their experiments, conducted on datasets like English Gigaword and DUC2004, demonstrated promising results. However, this approach, solely dependent on attention, struggled to effectively capture long-distance semantic relations, performing better with shorter sentences, Nallapati et al. [30]devised an innovative approach by substituting the encoder with an RNN in their encoder-decoder model, tailored specifically for abstractive text summarization tasks on both individual sentences and longer document collections. Alongside this, they introduced the CNN/Daily Mail dataset, notable for containing multi-sentence summaries. Recognizing the heightened complexity of this dataset, they enhanced their model's capacity to capture longer dependencies by integrating a pointer mechanism. This mechanism facilitated decision-making during decoding, enabling the model to choose between generating new words or selecting existing ones from the source document, thereby refining the accuracy of the generated summaries, Semantic data transformation was integrated with deep learning methodologies for application in abstractive summarization [31]. Initially, a general summary was crafted utilizing an encoder-decoder deep learning model. Subsequently, this summary underwent a transformation process to render it into a format easily readable by humans.

Kahla M.et al. [32] present the first corpus of human-written abstractive news summaries in Arabic; the dataset consists of more than 21 thousand items. They used this dataset to train a set of neural abstractive summarization systems for Arabic by fine-tuning pre-trained language models such as multilingual BERT, AraBERT, and multilingual BART-50. As the Arabic dataset is much smaller than e.g. the CNN/Daily Mail dataset, they also

applied cross-lingual knowledge transfer to significantly improve the performance of their systems. The setups included two M-BERT-based summarization models originally trained for Hungarian/English and a similar system based on M-BART-50 originally trained for Russian that were further fine-tuned for Arabic. Evaluation of the models was performed in terms of ROUGE, and a manual evaluation of fluency and adequacy of the models was also performed. Also Qusai I. et al. [33] employed a multilingual pre-trained model, t5-arabic-base, customized for both Arabic and English tasks such as summarization. They fine-tuned this model on news articles sourced from Aljazeera.net (AD), available on Kaggle, comprising 5870 articles in Arabic. Subsequently, they evaluated their model's performance using Precision, Recall, and F1-score metrics for ROUGE-1, ROUGE-2, and ROUGE-L.

1.8.2 Extractive Arabic Text Summarization:

AlSanie et al. [34] proposed one of the earliest Arabic text summarization systems based on Rhetorical Structure Theory (RST). The approach involves constructing Rhetorical Structure trees (RST-trees) to represent the structural organization of the source text by analyzing relationships between text segments. They utilized eleven Arabic rhetorical relations and twenty-five cue sentences for this purpose. The system generates summaries by selecting the most suitable tree. To evaluate its effectiveness, the authors created a corpus comprising various sources such as technical articles, newspaper articles, and books, covering diverse fields like accounting, technology, and society. Their system demonstrated promising results, particularly for small and medium-sized documents.

One of the earlier Arabic text summarization adopting an extractive methodology was LAKHAS. It was developed by Douzidia and Lapalme [35] it focuses on journalistic texts and leverages various statistical features such as sentence position, term frequency, title words, and cue words. The system's performance was evaluated by participating in the DUC 2004 (Document Understanding Conferences), where summaries were translated into English for assessment using the ROUGE measure.

Al-Radaideh and Bataineh [36] created a hybrid method for text summarization, employing an extraction approach. Their method integrates domain expertise, statistical features, semantic similarity, and genetic algorithms. Genetic algorithms are applied to select the best sentence combination for a summary, aiming to maximize both informative content and coherence. The method underwent testing on two datasets: the KALIMAT corpus and the Essex Arabic Summaries Corpus (EASC), with performance evaluation conducted using ROUGE and F-measure metrics. Ibrahim et al. [37] introduced a hybrid extractive Arabic text

summarization system that integrates Rhetorical Structure Theory (RST) and the Vector Space Model (VSM) to create summaries for Arabic text.

Many researchers have employed supervised techniques such as Markov hidden models, Bayesian approaches, and Support Vector Machines (SVM) for Automatic Text Summarization (ATS).

In article [38] the authors combine Bayesian and genetic programming (GP) classification techniques to produce summaries, employing a limited set of features. They train the system using manually annotated datasets. Results indicate that the Bayesian classifier typically exhibits high recall, while the GP classifier tends to have high precision, combining both classifiers leads to increased recall and summary size, whereas using the intersection of the two classifiers enhances precision while reducing the summary size.

The authors [39] used a graph based on the cosine similarity to construct it, and for the representation of the sentences they used the TF-IDF. For the evaluation they used an Arabic corpus. In [40]the authors proposed a graph-based method for Arabic text summarization. They noted the significance of nouns in Arabic sentences, as they typically convey more information. Therefore, their approach focuses on the frequency of nouns in sentences and utilizes a modified PageRank algorithm. Unlike the original PageRank, their method incorporates edge weights between nodes using cosine similarity. Additionally, instead of initializing the rank with 1/N as in the original PageRank, their algorithm begins with the number of nouns as the initial rank.

After examining related works on Arabic text summarization, it is clear that these studies have given us a broad understanding of the techniques and models used in this field. This knowledge will guide the development of our Deep Learning model for text summarization.

1.9 Conclusion:

In this chapter, we have explored the fundamental concepts and terminology central to our theme, which revolves around automatic text summarization. Beginning with an overview of Artificial intelligence and natural language processing and its diverse applications, we then delved into the specifics of text summarization, considering its relevance and challenges within the context of the Arabic language. Additionally, we conducted a comprehensive review of related works in the field, providing insights into various approaches and methodologies employed by researchers.

In the next chapter, we will explore deep learning concepts as well as techniques and models relevant to text summarization.

CHAPTER II:

Deep Learning and Text Summarization.

2.1 Introduction:

This chapter delves deeper into the transformative role of deep learning in advancing automatic text summarization, with a focus on applications specific to the Arabic language.
Deep learning, with its ability to model complex patterns in data, has revolutionized many areas of artificial intelligence, including NLP. This chapter will explore the evolution from traditional neural networks to sophisticated deep learning architectures that have significantly enhanced the capabilities of text summarization systems. We will examine the core concepts and developments, starting with basic Artificial Neural Networks (ANNs) and progressing through to advanced models such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and their variants like Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs).

Furthermore, we will explore the architecture and functionality of modern neural network designs such as the Encoder-Decoder model, Sequence-to-Sequence models, and the Attention Mechanism, which have been pivotal in handling the complexities of sequential data. The chapter will also provide an in-depth analysis of the Transformer architecture, a breakthrough in NLP that has laid the groundwork for state-of-the-art models in text processing.

A significant portion of this chapter is dedicated to pretrained models that have set new benchmarks in NLP tasks, including text summarization. We will discuss the workings and applications of models like BERT (Bidirectional Encoder Representations from Transformers), BART (Bidirectional and Auto-Regressive Transformers), and T5 (Text-To-Text Transfer Transformer), exploring how these models have been adapted for summarization tasks.

2.2 Artificial Neural Network (ANN):

An Artificial Neural Network (ANN) is an information-processing paradigm inspired by the way biological nervous systems, such as the brain, process information. [41]
An artificial neural network comprises processing units known as neurons, which aim to replicate the structure and behavior of natural neurons. Each neuron consists of inputs (dendrites) and one output (synapse via axon), with a function that governs the neuron's activation.

x1...xn represent the inputs to the neuron, with a bias added to the neuron along with the inputs, typically initialized to 1. W0...Wn denote the weights, where each weight corresponds to a connection to the signal. The strength of the signal is determined by the product of the weight and the input. While a neuron receives multiple inputs from different sources, it produces a single output. Activation functions vary, with one of the most commonly used being the sigmoid function. [42]

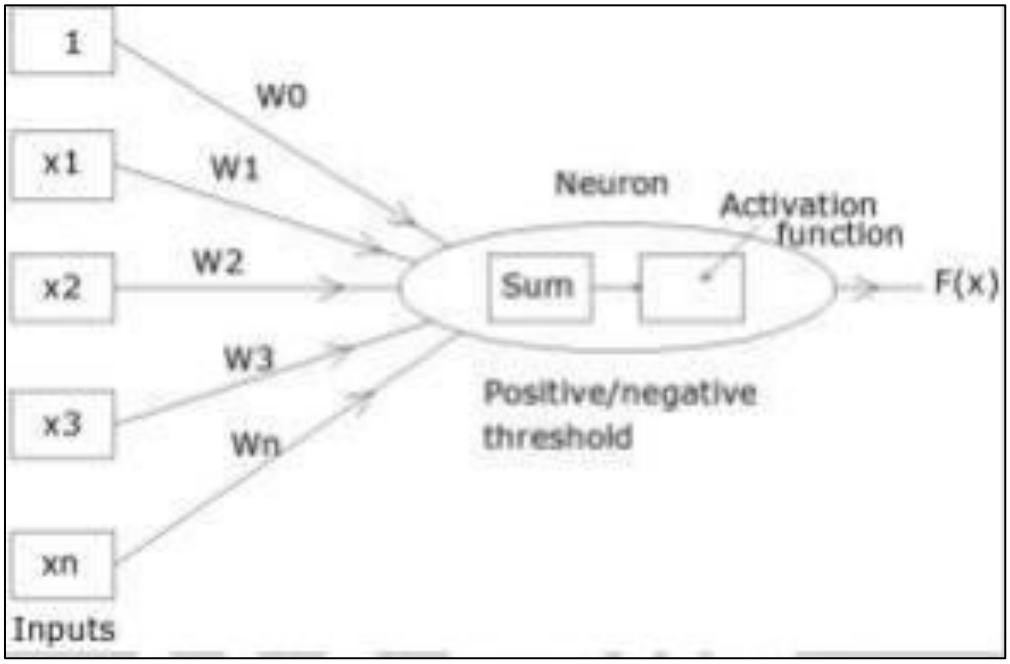


Figure 2.1: Model of an artificial neuron. [43]

The ANN architecture comprises of:

1. Input layer: Receives the input values.
2. Hidden layer(s): A set of neurons between input and output layers. There can be single or multiple layers.
3. Output layer: Usually it has one neuron, and its output ranges between 0 and 1, that is, greater than 0 and less than 1. But multiple outputs can also be present. [42]

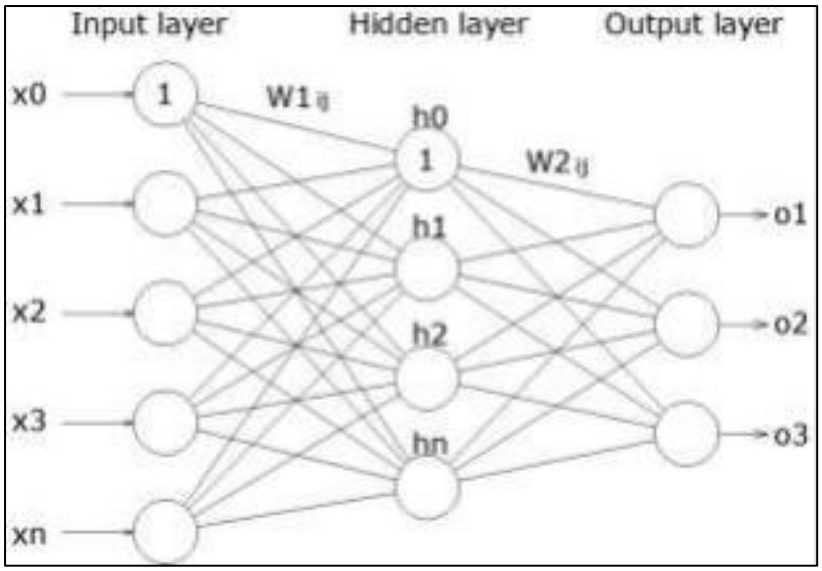


Figure 2.2: Neural network architecture [43]

2.3 Deep learning:

Deep Learning (DL) has become a prominent topic in artificial intelligence discussions today. However, there is no single definition of DL in literature. Some define it as: "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" [44]. Others describe it as a process not only learning relationships between variables, but also the knowledge governing those relationships and making sense of them [45]. This definition is not limited to hierarchical abstractions, representations, or concepts, but extends to networks as well.

2.4 Deep Neural Networks (DNN) :

Deep learning specifically refers to deep neural networks, organized into multiple consecutive layers. While they are still neural networks, deep networks typically have more layers. Their distinguishing feature is their ability to address problems that were previously inaccessible to traditional methods and smaller neural networks.

2.5 Deep Neural Networks (DNN) architectures:

2.5.1 Convolutional neural network CNN:

A convolutional neural network or CNN is a special kind of artificial neural network that simulates the human visual behavior. It plays a very peculiar role in computer vision. The concept was introduced by Yann LeCun and Yoshua Bengio in 1995. Convolution is a specialized linear operation, commonly used in convolutional neural networks (CNNs). These networks, similar to traditional neural networks, consist of neurons with adjustable weights and biases. Each neuron receives inputs, performs a dot product, and may apply nonlinearity operations such as sigmoid, tanh, ReLU, or softplus. CNNs are structured with various layers

including convolutional layers, which use kernels for convolution operations, nonlinearity layers, pooling layers, flattening, and fully connected layers. [46]

Typical convolutional neural network architecture is shown in **Figure 2.3**:

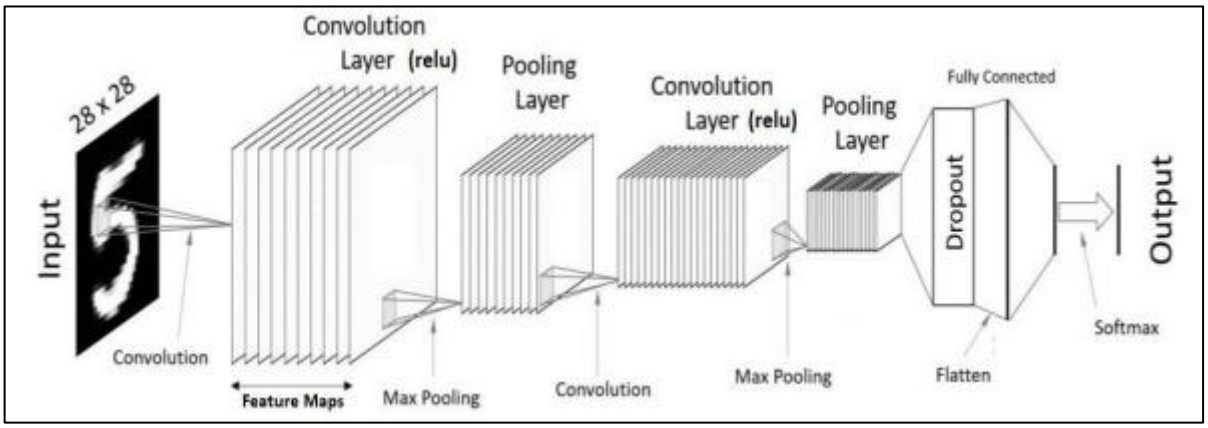


Figure 2.3: Architecture of a typical convolutional neural network. [46]

2.5.2 Recurrent neural networks (RNN):

Recurrent Neural Network is a type of network characterized by loops, enabling the persistence of information within the network. Through feedback connections to itself, activations can circulate in a loop, facilitating the learning of sequences and the retention of information. RNNs exhibit remarkable prowess in modeling sequential data, such as speech or text, and can even be applied to non-sequential data for training in a non-linear manner. Their versatility extends to various applications, including image and video captioning, word prediction, translation, image processing, speech recognition, natural language processing, and music processing, etc. [47]

A basic RNN comprises three layers: input, recurrent hidden and output layers, as depicted in Figure a. The input layer consists of N input units. Inputs to this layer are sequential vectors over time t, denoted as {..., xt−1, xt, xt+1, ...}, where xt = (x1, x2, ..., xN). In a fully connected RNN, input units are linked to hidden units in the hidden layer via connections defined by a weight matrix WIH. The hidden layer comprises M hidden units ht = (h1, h2, ..., hM), interconnected across time through recurrent connections, as shown in **Figure 2.4**. [48]

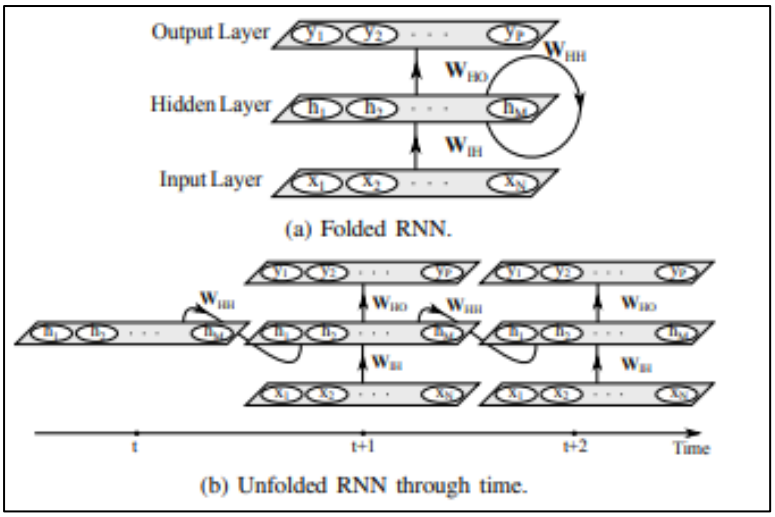
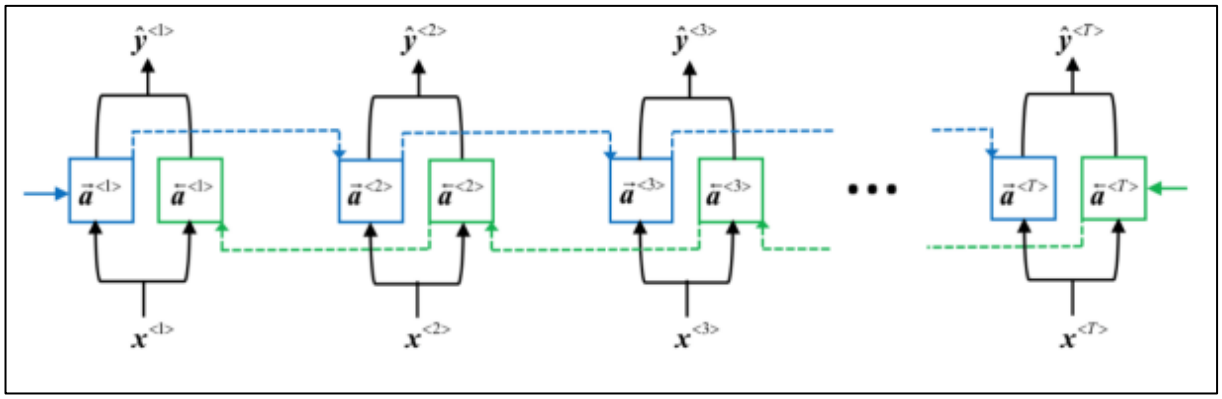


Figure 2.4: A simple recurrent neural network (RNN) [48]

2.5.3 Bidirectional recurrent neural networks (BRNN):

In 1997, Schuster and Paliwal [49] introduced a bidirectional recurrent neural network (BRNN) as a solution to the limitations of traditional (unidirectional) RNNs. Unlike unidirectional RNNs, BRNNs utilize information from both past and future time steps to generate output at a given time step. This is achieved by training the network in both the positive and negative directions simultaneously. In this setup, neurons in the network are divided into two parts: one responsible for processing information in the positive direction and the other for the negative direction. Outputs from neurons in one direction are not connected to neurons in the opposite direction [50], resulting in the structural arrangement



illustrated in **Figure 2.5**.

Figure 2.5: A bidirectional sequence learning model with one hidden layer in the unfolded form. [50]

2.5.4 Long Short-Term Memory (LSTM):

Long Short-Term Memory (LSTM) is a specific recurrent neural network (RNN) architecture that was designed to model temporal sequences and their long-range dependencies more accurately than conventional RNNs [51]. Within the LSTM architecture, there exist specialized units known as memory blocks situated in the recurrent hidden layer. These memory blocks are comprised of memory cells with self-connections, which store the temporal state of the network. Additionally, they feature multiplicative units referred to as gates, which regulate the flow of information. In the original design, each memory block included both an input gate and an output gate. The input gate manages the influx of input activations into the memory cell, while the output gate governs the release of cell activations into the broader network. [51]

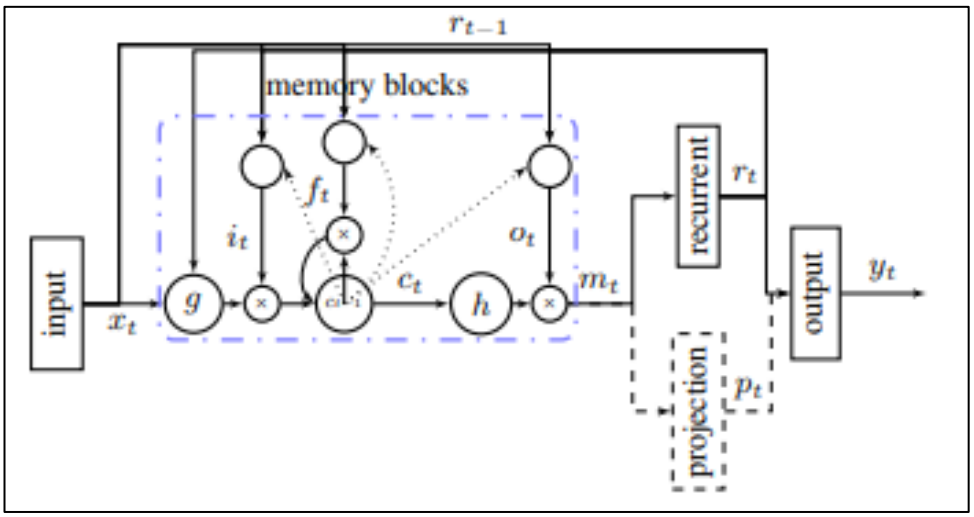


Figure 2.6: LSTM based RNN architectures with a recurrent projection layer and an optional non-recurrent projection layer. [51]

2.5.5 Gated recurrent units (GRUs):

In 2014, Cho et al. [52] and Chung et al. [53] introduced the Gated Recurrent Unit (GRU) as a simpler alternative to the LSTM. Compared to the LSTM, the GRU has a simpler internal structure, making it easier to train due to fewer computations involved. These simplifications are primarily achieved in two ways: 1/ Combining the input and forget gates into a single gate known as the update gate, and 2/ Merging the internal/cell state with the hidden state [50], the **Figure 2.7** shows the GRU unit cell:

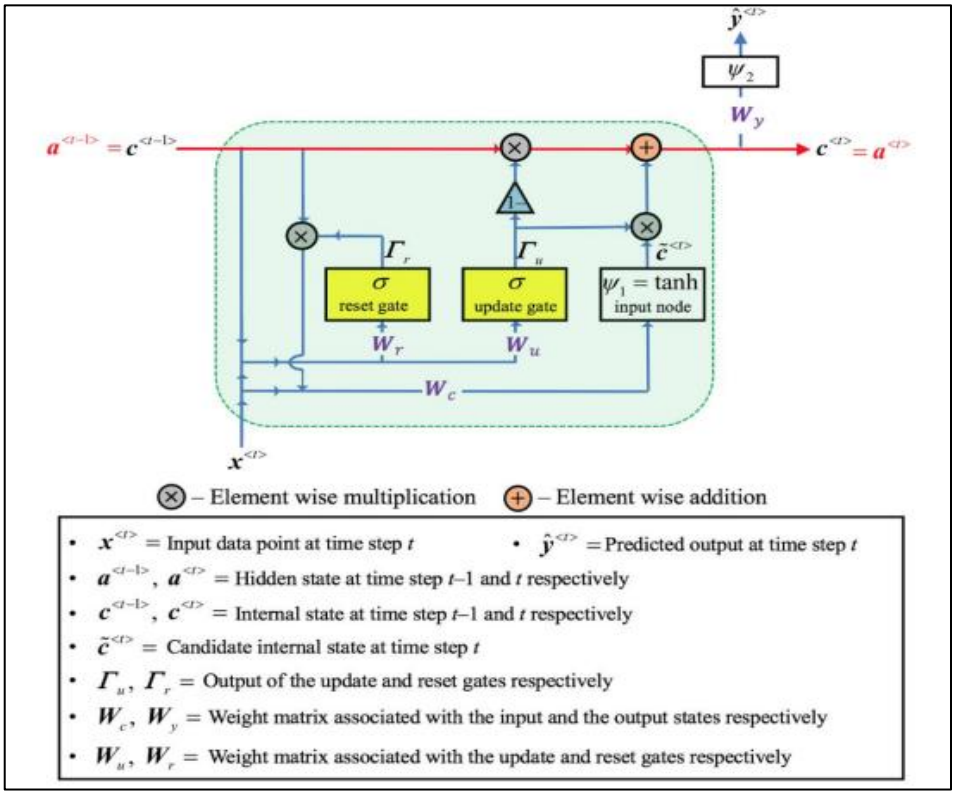


Figure 2.7: GRU unit cell (repeating module in a GRU network). [50]

2.6 Encoder-Decoder:

Introduced by Sutskever et al. in 2014 [54]. This type of neural network consists of two main components: an encoder and a decoder. The decoder can be a recurrent neural network (RNN) or one of its variants, such as Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRU), to enhance system performance.

The encoder's role is to process the input text sequence and convert the relevant information into a context vector (encoder vector). After traversing the entire sequence, the encoder passes this context vector to the decoder. The decoder then uses the context vector to generate the output sequence accurately.

One significant advantage of this model is its ability to handle input sequences of variable lengths and produce output sequences of variable lengths. This capability facilitates applications such as text summarization and text translation.

The **Figure 2.8** illustrates the architecture of the Encoder-Decoder model.

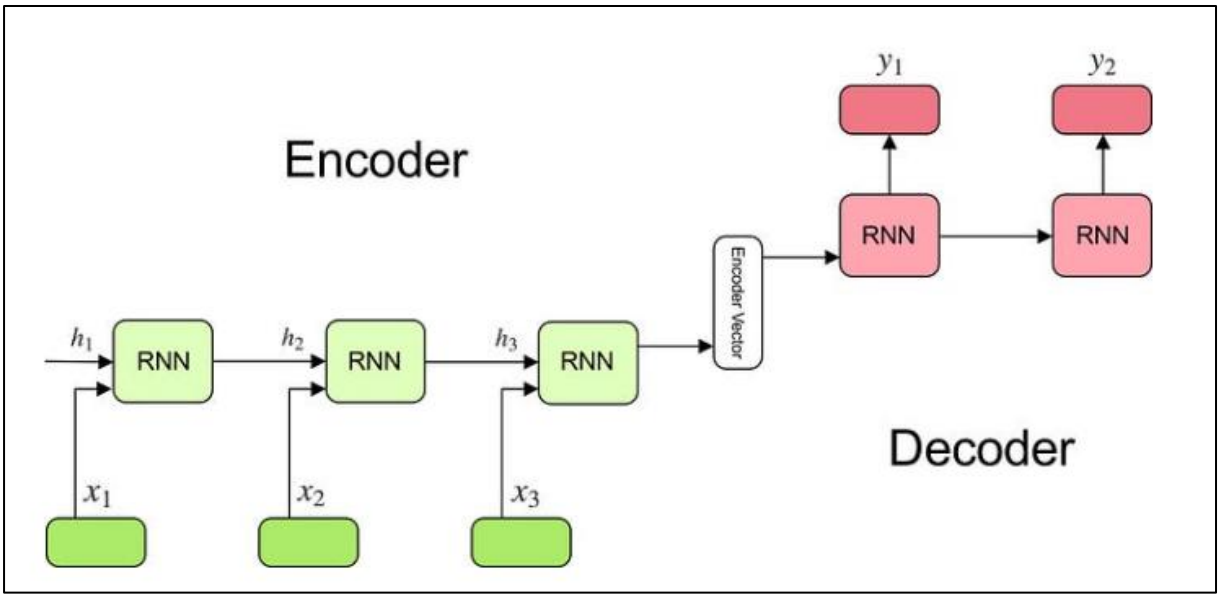


Figure 2.8: Architecture of Encoder Decoder [55]

2.7 Sequence-to-sequence models:

Although DNNs offer versatility and power, their applicability is limited to tasks where inputs and outputs can be effectively encoded as fixed-dimensional vectors. They are unsuitable for scenarios with variable sequence lengths, such as text summarization, where the input consists of a lengthy string of words and the output is also a sequence.

The concept involved employing an LSTM network to process the input sequence step by step, generating a sizable, fixed-dimensional vector representation. Subsequently, another LSTM was utilized to decode the output sequence from this vector. This approach is known as the sequence-to-sequence model. [54]

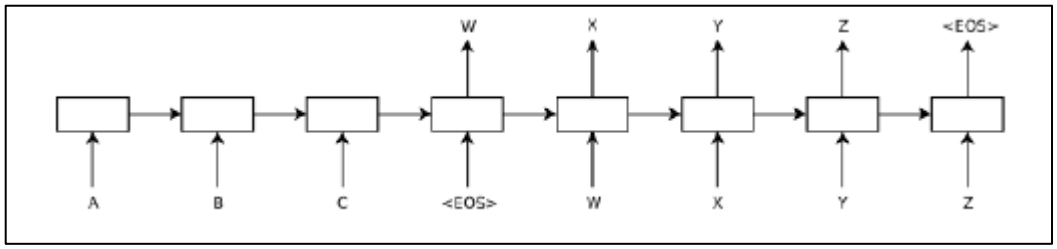


Figure 2.9: Sequence to sequence architecture [54]

A seq2seq setup for text summarization consists of two main components: an encoder and a decoder. The encoder processes a source article, represented by x = (x1, x2,..., xJ), and converts it into hidden states he = (he1 ,he2 ,...,heJ). Subsequently, the decoder utilizes these hidden states as contextual input to generate a summary y = (y1,y2,...,yT). In this setup, xi and yj represent one-hot encoded tokens from the source article and summary, respectively.

Encoders and decoders can be feed-forward networks, CNN or RNN. RNN architectures, especially long short-term memory (LSTM) and gated recurrent unit (GRU), have been most widely adopted for seq2seq models. **Figure 2.10** shows a basic RNN seq2seq model with a bidirectional LSTM encoder and an LSTM decoder. The bi-directional LSTM is considered since it usually gives better document representations compared to a forward LSTM. [56]

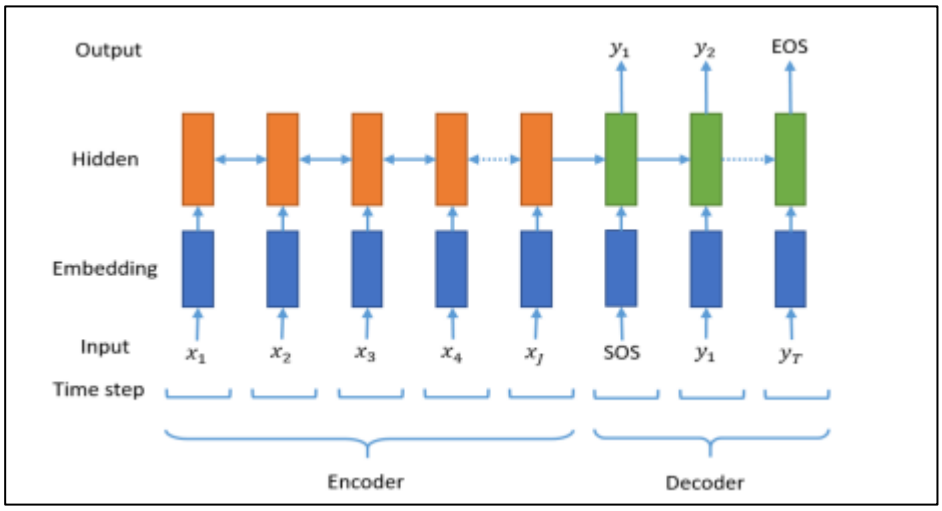


Figure 2.10: The basic seq2seq model. SOS and EOS represent the start and end of a sequence, respectively [56]

2.8 Attention mechanism:

The attention mechanism was initially used for neural machine translation before being applied to NLP tasks like text summarization. In a basic encoder-decoder architecture, handling long sentences can be problematic because the encoding size is fixed for the input string, preventing it from accounting for all elements of a lengthy input. To address this, the attention mechanism was introduced, allowing the model to focus on significant parts of the input when generating a summary. The attention mechanism calculates weights between each output word and every input word at each output step, ensuring the weights sum to one. These weights indicate which input words should be emphasized for each output word. The weighted average of the decoder's last hidden layers at the current step is then computed for each input word and fed into the softmax layer along with the last hidden layers. [57]

2.9 Transformers:

The Transformer model, introduced by Vaswani et al. [58] , has demonstrated strong performance across various tasks, including machine translation and language modeling. With the emergence of pretrained language models, Transformer architectures have gained even more prominence, serving as the foundation for many state-of-art natural language processing models.

A Transformer model comprises multiple layers, or blocks, each consisting of a self-attention module followed by a feed-forward layer. Additionally, layer normalization and residual connections are integral components of each layer. [59]

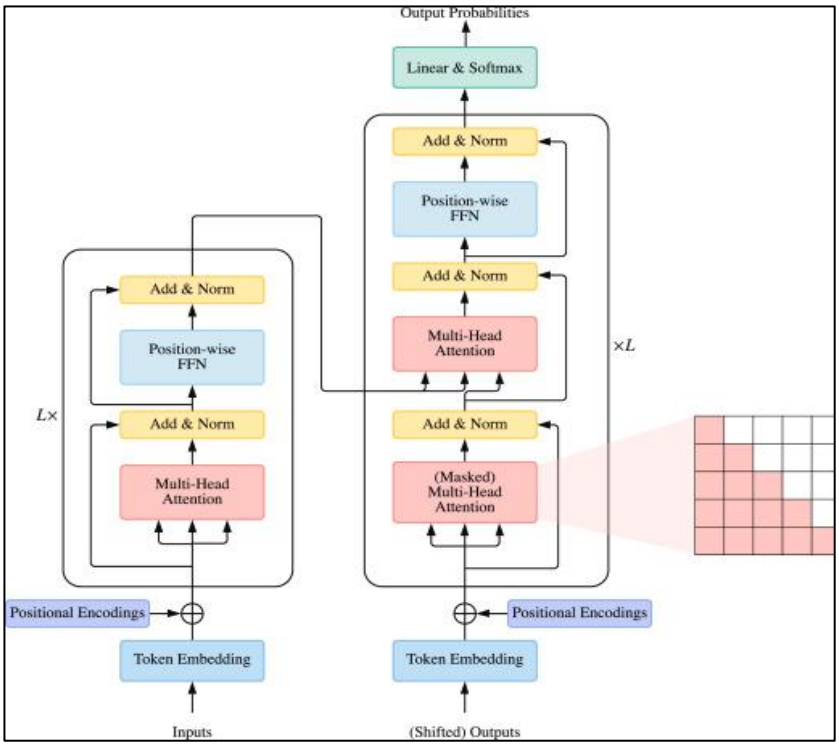


Figure 2.11: Overview of Transformer architecture. [60]

2.10 Pre-trained models Based on Transformers:

2.10.1 BERT (Bidirectional Encoder Representations from Transformers):

BERT [61] is among the leading methods in NLP. It utilizes a stack of transformer encoders, constructed from transformer encoder blocks. It remains one of the top models for various NLP tasks such as question answering, natural language understanding, sentiment analysis, and text summarization.

BERT employs Masked Language Modeling (MLM), where 15% of the words in the input document/corpus are masked. The model is then tasked with predicting these missing words. This process involves presenting sentences with masked words to each transformer encoding block in the stack, which then predicts the missing word. Since any word can be masked in a sentence and the model must predict it, BERT operates in an unsupervised manner. This means it doesn't require labeled data, allowing it to leverage vast amounts of unlabeled text data from the internet for training. This process is illustrated in the **Figure 2.12** below. [62]

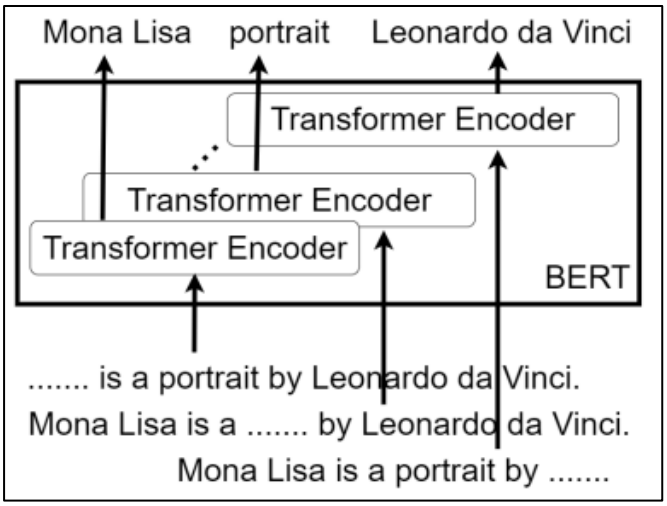


Figure 2.12: Feeding sentences with missing words to the BERT model for training. [62]

BERT learns to predict missing words by considering both preceding and succeeding words, making it bidirectional. This means it takes into account the context of each word from both directions. Unlike word2vec and GloVe, which offer a single embedding for each word regardless of context, BERT provides context-aware embeddings. This means that the embedding of a word can vary depending on its context within different sentences. For instance, the word "bank" would have different embeddings in the sentences "Money is in the bank" and "Some plants grow in bank of rivers" due to its different meanings in each context.

Aside from being trained on the Masked Language Model (MLM), BERT underwent pre-training via an unsupervised task known as **Next Sentence Prediction (NSP).** This was to enable the model to understand the relationship between two sentences. During NSP, sentences were generated from any monolingual text corpus, and then pairs of sentences A and B were selected: half the time, B was the immediate sentence following A, labeled as IsNextSentence, while the other half of the time, it was a randomly chosen sentence from the corpus, labeled as NotNextSentence. [61]

In short, BERT was pre-trained on vast text datasets using two unsupervised tasks: MLM and NSP. It employs a large model with 12 layers for BERTbase and 24 layers for BERTlarge. To create high-performance models with BERT, fine-tuning involves adding just one output layer and updating pre-trained parameters with labeled data from specific tasks. [61]

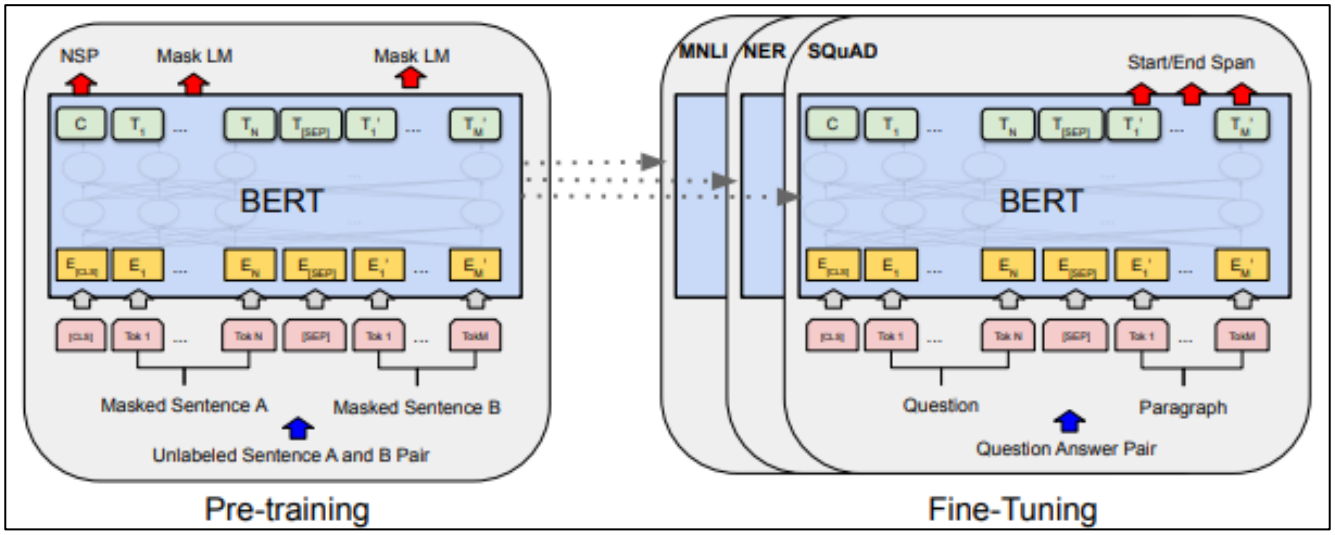


Figure 2.13: Overall pre-training and fine-tuning procedures for BERT. [61]

Figure 2.13 illustrates that, besides the output layers, the same architectures are employed for both pre-training and fine-tuning. The pre-trained model parameters are utilized to initialize models for various down-stream tasks. During fine-tuning, all parameters are adjusted. "[CLS]" signifies a unique symbol appended at the start of each input example, while "[SEP]" serves as a special separator token. [61]

2.10.1.1 BERT models:

As mentioned earlier, BERT underwent training on extensive text datasets, resulting in two models: BERT-base with 12 layers transformers and BERT-large with 24 layers. Subsequently, several popular models emerged for various languages, including English and Arabic. Some of them are listed below:

- **RoBERTa model:**

RoBERTa (Robustly Optimized BERT pre-training Approach), a variant of BERT, aims to enhance BERT by adjusting various parameters. It explores the impact of larger pre-training datasets, dynamic masking, batch sizes, and text encoding methods. Trained on 160GB of uncompressed English text from various sources, RoBERTa utilizes dynamic masking, large mini-batches, and BPE encoding. While NSP's necessity is investigated, its removal does not significantly affect performance. RoBERTa's performance surpasses XLNet

and BERT in NLP tasks, but it requires extensive data and computational resources, leading to increased complexity and execution time. [63]

- **DistilBERT model**:

Distillation in neural networks, proposed by [64], aims to speed up models by replacing larger architectures with lighter versions. DistilBERT follows this approach, reducing the number of layers in BERT-base by half and eliminating token embeddings and poolers. Despite using dynamic masking and omitting next sentence predictions, DistilBERT maintains 97% of BERT's performance with 40% fewer parameters, and it's 60% faster. Evaluated on the GLUE benchmark, it performs well on downstream tasks. While DistilBERT excels in speed and efficiency, it inherits BERT's fixed-length limitation. [63]

- **AraBERT model:**

AraBERT is a widely used model in various NLP tasks for several languages, it is a pre-trained model for the Arabic language, based on the Google BERT architecture, The model comes in six versions: AraBERTv0.1-base, AraBERTv0.2-base, AraBERTv0.2-large, AraBERTv1-base, AraBERTv2-base, and AraBERTv2-large. **Table 2.1** outlines the key details of each version regarding the pre-training procedure. [65]

Model	Size		Pre-segmentation	Dataset		
	MB	Param.		Sentences	Size	Words
AraBERTv0.2-base	543 M	136M	No	200 M	77 GB	8.6 B
AraBERTv0.2-large	1.38 G	371M	No	200 M	77 GB	8.6 B
AraBERTv2-base	543 MB	136M	Yes	200 M	77 GB	8.6 B
AraBERTv2-large	1.38 G	371M	Yes	200 M	77 GB	8.6 B
AraBERTv0.1-base	543 MB	136M	No	77 M	23 GB	2.7 B
AraBERTv1-base	543 MB	136M	Yes	77 M	23 GB	2.7 B

Table 2.1: Different version of AraBERT. [65]

2.10.2 GPT model:

GPT (Generative Pre-trained Transformer) is a deep learning model produced by OpenAI, undergoes pre-training on extensive textual datasets. It's adaptable for various tasks including language generation, sentiment analysis, and translation and text classification. Employing transformer architecture, GPT surpasses older NLP techniques like RNN and CNN, It uses a self-attention mechanism to allow the model to consider the context of the entire sentence when generating the next word, which improved the model’s ability to understand and generate language. [66]

2.10.3 BART model:

BART is a model combining Bidirectional and Auto-Regressive Transformers, operates as a denoising auto-encoder, Pretraining has two stages: first: text is corrupted with an arbitrary noising function, and second: a sequence-to-sequence model is learned to reconstruct the original text. BART employs a standard Transformer-based architecture, akin to a fusion of BERT (with its bidirectional encoder) and GPT (utilizing a left-to-right decoder).

In **Figure 2.14A**: Random tokens are substituted with masks, and the document undergoes bidirectional encoding. Absent tokens are predicted individually, making BERT unsuitable for generation tasks. [67]

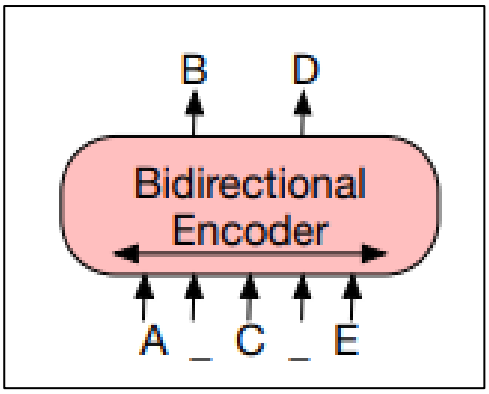


Figure 2.14A: BERT. [67]

In **Figure 2.14B**: Tokens are predicted sequentially, allowing GPT to be employed for generation tasks. However, it only considers leftward context for word conditioning, limiting its ability to learn bidirectional interactions.

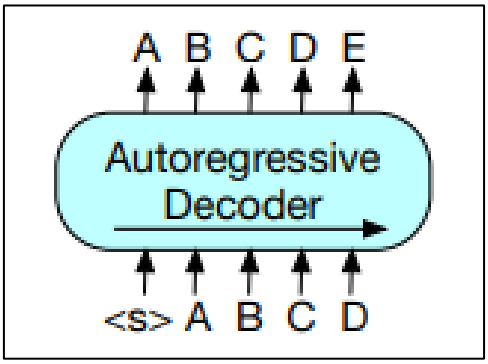


Figure 2.14B: GPT. [67]

In **Figure 2.14C**: The encoder doesn't require alignment with decoder outputs, enabling arbitrary noise alterations. In this scenario, sections of a document have been altered

by replacing them with mask symbols. The modified document is then encoded using a bidirectional model, and the probability of the original document is determined using an autoregressive decoder. During fine-tuning, an unaltered document is fed into both the encoder and decoder, utilizing representations from the decoder's final hidden state.

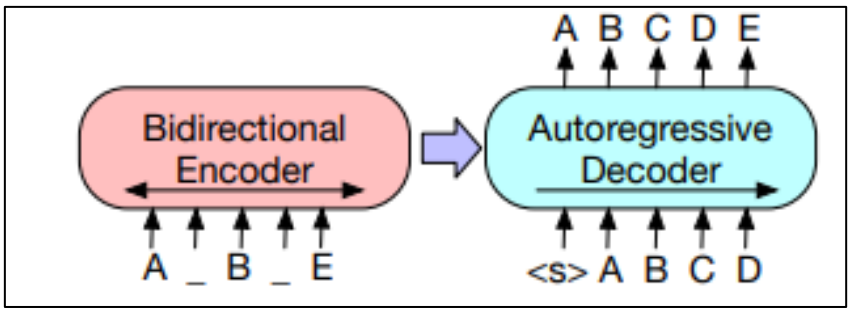


Figure 2.14 C: BART. [67]

BART's architecture shares similarities with BERT, but with a few distinctions:

First: Each decoder layer also conducts cross-attention over the encoder's final hidden layer, akin to the transformer sequence-to-sequence model.

Second: Unlike BERT, BART does not employ an extra feed-forward network before word prediction.

Overall, BART has about 10% more parameters than a comparably sized BERT model.

BART underwent training by corrupting text using a random obfuscation function, enabling the application of various document corruptions. The experiment utilized several modifications, including token masking, token deletion, text infilling, sentence permutation, and document rotation, as summarized below (**Figure 2.15**). [67]

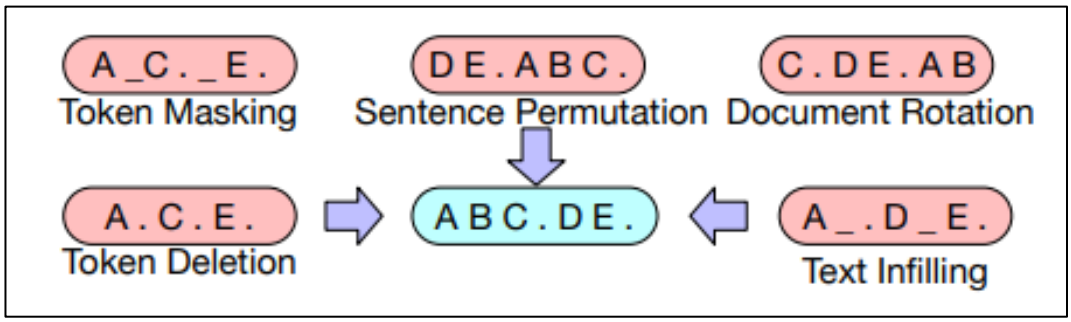
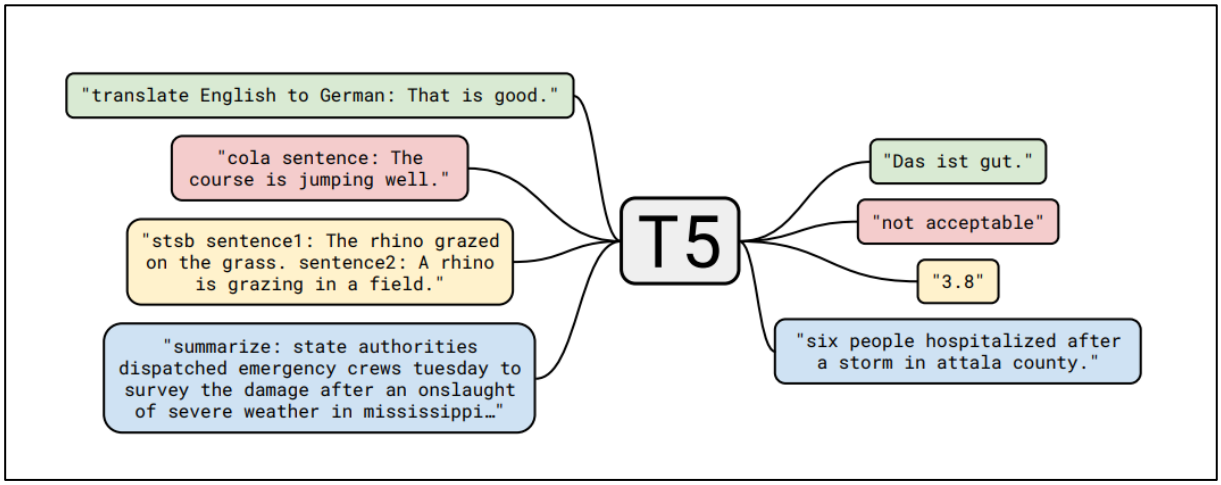


Figure 2.15: Transformations for noising the input that we experiment with. [67]

2.10.4 Text-To-Text Transfer Transformer (T5):

T5, which stands for Text-To-Text Transfer Transformer, is a model introduced by [68] . It is built on the original encoder-decoder transformer architecture.

The peculiarity of the T5 model is that it converts all NLP problems into identical text-to-text format, the input and output of the model are always texts, and this allows the model to perform any NLP task with the same model, the same cost function and the same parameters at the same time. To specify which task should be performed by the T5 model, it is enough to



add a prefix to the model input. The figure illustrates the multitasking concept of the T5 model

Figure 2.16: multi-tasking concept of the T5 model. [68]

Unlike other popular transformer-based models that are typically used for classification tasks, T5 treats all NLP problems as text-to-text operations, meaning both the input and output are text strings. While this approach seems especially suited to tasks like question answering, translation, and summarization, it can be applied to a wide range of NLP problems. [69]

2.11 Conclusion :

This chapter underscored the significant impact of deep learning methodologies on text summarization, particularly for the Arabic language. The integration of these advanced models and architectures has demonstrated potential in making summarization systems more effective and adaptable, opening new possibilities for their practical applications.

As we conclude this chapter, it is clear that deep learning has fundamentally reshaped the landscape of automatic text summarization. The insights provided here lay a solid groundwork for the next chapter, where we will delve into our proposed approach and methodology for Arabic automatic text summarization. We will explore the detailed design, implementation strategies, and the innovative techniques employed to further advance the field.

CHAPTER II: Deep Learning and Text Summarization.

CHAPTER III :

MODEL CONCEPTION

CHAPTER III: MODEL CONCEPTION

3.1 Introduction:

The Arabic language, with its complex morphology and diverse dialects, poses significant challenges for natural language processing (NLP). Effective text summarization in Arabic requires advanced models capable of handling these intricacies.

Recent advances in deep learning have led to powerful models like BART (Bidirectional and Auto-Regressive Transformers) which have shown remarkable performance in various text generation tasks.

This chapter outlines the preparatory steps for developing an automatic Arabic text summarization based on mBART model. It covers the proposed approach, methodology, and details of dataset preparation and preprocessing. Additionally, it delves into the process of fine-tuning the model to adapt it specifically for Arabic text summarization, addressing the unique linguistic characteristics of the language.

3.2 Methodology:

In this part, we present details of the methodology, the process followed in the design of our arabic text summarizer composed of 3 stages:

- Loading the dataset
- Training of the model on the training dataset (Training-set), and the validation of results on the validation set (Validation set),
- Evaluate the performance of the model on the test set (Test-set).

Each step is reviewed in more depth in the following sections. Fig. 3.1 next illustrates the processes.

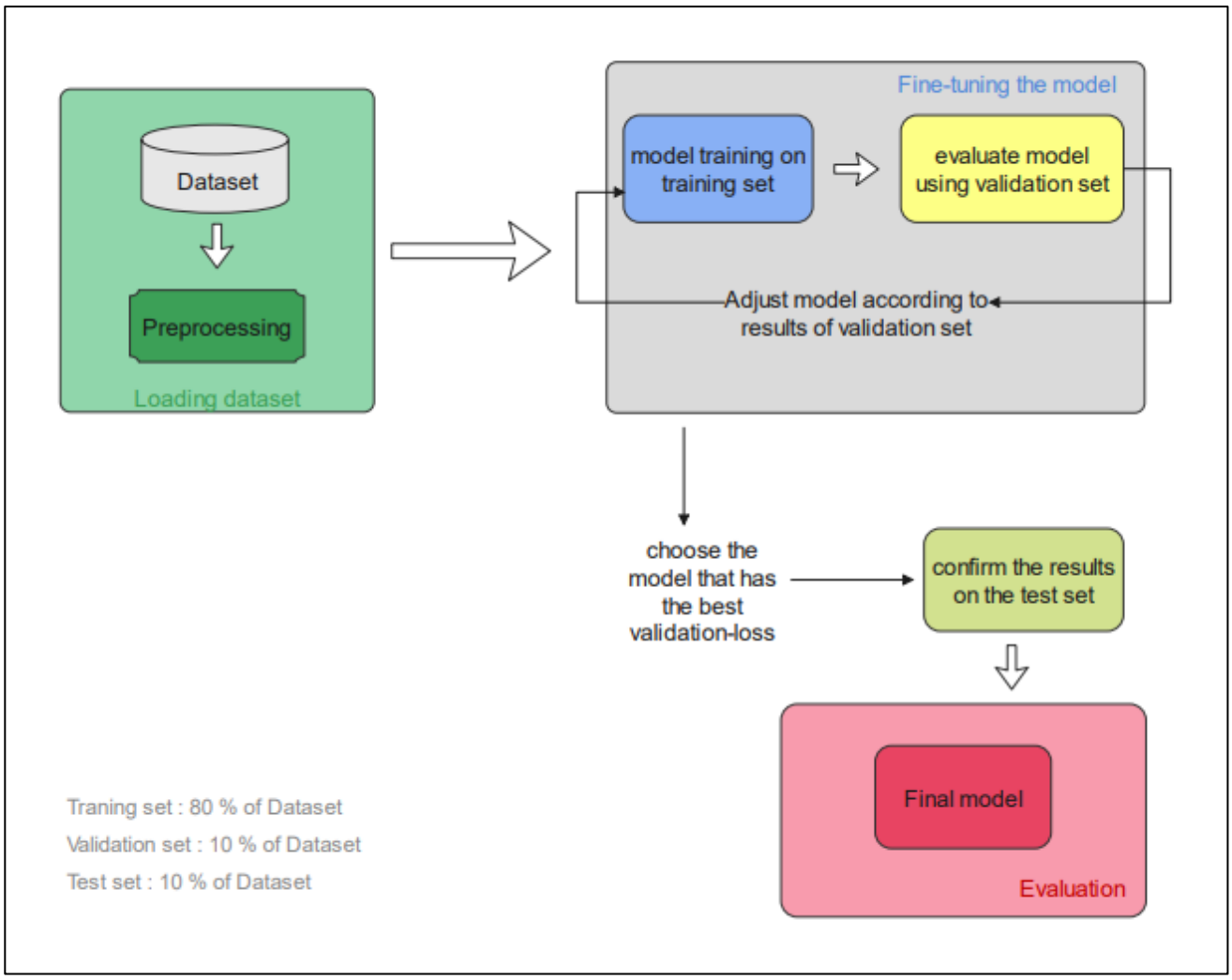


Figure 3.1: the methodology

3.2.1 Dataset:

3.2.1.1 Dataset Description:

The dataset used in this work is the XLSum [70], which is:

- an extensive and diverse dataset
- Consists of 1 million professionally annotated article-summary pairs from BBC content extracted using meticulously designed heuristics.
- spans 44 languages, including both low- and high-resource languages—many of which lack any public datasets— for our research to develop an Arabic text summarizer
- The newer version of the dataset contains a total of 1.35 million article-summary pairs and covers 45 languages.

- The high quality, abstractive, and concise nature of the summaries, as confirmed by human and intrinsic evaluations, provided a robust foundation for training and evaluating our summarization model, enabling us to create a sophisticated and reliable Arabic text summarizer.

3.2.1.2 Dataset Structure:

Data Fields:

- **'id':** A string representing the article ID.
- **'url':** A string representing the article URL.
- **'title':** A string containing the article title.
- **'summary':** A string containing the article summary.
- **'text' :** A string containing the article text.

Data Instances:

One example from the Arabic dataset is given below in JSON format:

{'id': '121115_israel_gaza_shelling',

'url':'https://www.bbc.com/arabic/multimedia/2012/11/121115_israel_gaza_shelling',

'title': 'مقتل ثلاثة إسرائيليين في قصف صاروخي من قطاع غزة',

'summary': ' افادت مصادر حركة حماس أن كتائب القسام، الجناح العسكري لحماس، أطلقت صاروخاً من طراز فجر خمسة على مدينة تل أبيب.',

'text': ' وكانت اذاعة الجيش الاسرائيلي افادت بمقتل ثلاثة اسرائيليين واصابة خمسة اخرين صباح اليوم نتيجة سقوط صواريخ فلسطينية. كما ارتفع عدد القتلى الفلسطينيين الى ثلاثة عشر شخصاً واصيب نحو مائة واربعين اخرين في الغارات الاسرائيلية على قطاع غزة. وكانت الطائرات الاسرائيلية شنت غارات على منطقة جباليا شمال القطاع وطاولت مخيم البريج وحي الزيتون جنوب القطاع.'}

3.2.1.3 Preprocessing the dataset:

Data preprocessing is an essential first step in building a deep Learning model, this step can refer to the deletion of data that can negatively affect the performance of the model or to the cleaning and preparation of data before they are used.

In order to guarantee the quality of the data used in our model, we have followed the following steps:

A- Cleaning the dataset:

This step consists of converting the textual data into a coherent format so that the model can understand the language. This step comprises the following operations:

- **Removing Empty and Duplicate Rows:**

Empty rows provide no useful information and can confuse the model during training, removing them ensures that the dataset contains only relevant and useful data. Similarly, Duplicate rows introduce redundancy and may bias the model, by removing duplicates; we ensure that each unique text contributes equally to the model's learning process, improving the generalizability of the model.

- **Removing diacritics:**

In Arabic, diacritics (Tashkeel) are used to indicate short vowels and other phonetic features. While they are important for pronunciation and disambiguation in reading, they can introduce unnecessary complexity in the text for NLP tasks. Removing diacritics simplifies the text, making it easier for the model processing and learning.

- **Removing punctuation and Special Characters:**

punctuation and special characters (e.g., marks, symbols) often do not carry meaningful semantic content for text summarization tasks. Their presence can introduce noise and distract the model from the relevant linguistic features. Removing these elements helps in streamlining the text, making it more uniform and easier for the model to interpret and process.

- **Eliminating Common Misspellings:**

Arabic script includes several letters that are phonetically similar but visually distinct, such as the different forms of the letter "alif" (ا, آ, إ, أ). These variations can lead to inconsistencies and misspellings in the text. Standardizing these forms by converting them to a single character (e.g., converting all forms of "alif" to ا) can reduce variability in the dataset. This step helps in improving the consistency of the text and ensures that the model is not confused by different representations of the same sound.

- **Removing URLs:**

URLs, point to external resources and add clutter without contributing to the text's semantic value. By stripping away HTML tags and URLs, we ensure the dataset is cleaner and more focused on the main content.

B- Data Splitting:

This step consists of dividing our dataset into three subsets:

- Training set: it is a subset to train a model.
- Validation Set: This set is used to tune hyperparameters and make decisions about the model architecture.
- Test Set: This subset is used to evaluate the model's performance.

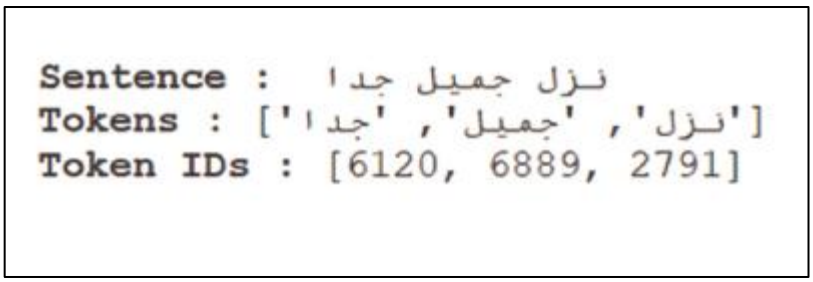
By allocating 80% of the data for training, we ensure that the model has ample data to learn from. The remaining 20% is equally divided between the validation and test sets, allowing for effective tuning and unbiased performance evaluation.

3.2.2 Tokenization:

In the case of deep 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 that can be sentences, words or characters. Then each token is transforms into a corresponding unique identifier in the pre-trained vocabulary (Input IDs) so that it can be used as entry for the model.

The tokenizer output comprises a dictionary with three essential elements:

- Input IDs: These represent the indexes of each token in the provided text.
- Attention Mask: It indicates which tokens should be attended to during processing.
- Labels: These are the indices associated with each token in the generated summary.



Tokenization is done as shown in Figure 3.2:

Figure 3.2: example of tokenization [71]

Tokenization in the Arabic language is a challenging task because of its rich morphology, in our case we load the associated pre-trained tokenizer of the model to guarantee that we obtain a tokenizer that matches the model architecture we are using.

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

3.2.3 Fin-tuning:

Fine-tuning the model involves re-training the model and updating its weights using the data specific to text summarization previously mentioned. This process requires careful consideration of several hyperparameters that control the training process,This adaptation aims to refine the model's parameters to better suit the nuances of the target task, in this case, Arabic text summarization.

3.2.4 Test:

Once the model has been trained and saved, the final step involves utilizing it to evaluate predictions for Arabic text summarization. For this purpose, a distinct set of Arabic summarization texts, known as the test set, is employed. This set is different from both the training and validation sets.

During the testing phase, the model processes the text from the test set and generates summaries based on its learned patterns and associations. These generated summaries are then compared against reference summaries to evaluate the quality of the model's predictions.

The evaluation metrics used on the test set provide valuable insights into the model's strengths and weaknesses. Common metrics for text summarization tasks include ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scores, which measure the overlap between the model-generated summaries and the reference summaries.

Ultimately, the performance of the model on the test set informs researchers and practitioners about its real-world applicability and effectiveness in Arabic text summarization tasks.

3.3 Proposed Approach:

In this work, we present a deep-learning model for the Abstractive automatic text summarization; our model leverages the BART (Bidirectional and Auto-Regressive Transformers) architecture, which combines the strengths of both BERT and GPT models. BART consists of a stack of encoders and decoders is built by the combination of multiple layers:

Encoder Layers:

The encoder is composed of several layers of transformers, Each transformer layer consists of a multi-head self-attention mechanism and a feed-forward neural network. Layer normalization are applied to stabilize training and help the model converge faster.

Multi-head attention Mechanism:

Self-Attention mechanism allows the model to weigh the importance of different tokens in the input sequence relative to each other. The self-attention mechanism can be

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \qquad (3.1)$$

defined as: [58]

Where Q, K, and V are the query, key, and value matrices derived from the input embeddings, and d□ is the dimension of the key vectors.

multi-head attention leverages multiple self-attention mechanisms to improve the model's ability to capture intricate patterns and dependencies in the data, it enables the model to simultaneously focus on information from various representation subspaces at different positions. This is something a single attention head cannot achieve, as it loses detail due to

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

averaging. [58]

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. (3. 3)

Feed-Forward Neural Network:

A two-layer fully connected feed-forward network with a GeLU activation in between, applied to each position separately and identically. It can be defined as: [58]

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \qquad (3.\ 4)$$

Where W1, W2, b1, and b2 are learned parameters.

Layer normalization:

Applied to the output of both the self-attention mechanism and the feed-forward network, typically following each sub-layer (with a residual connection added).
Layer normalization can be defined as a function mapping LN: $\mathbb{R}^D \to \mathbb{R}^D$ with two set of adaptive parameters, gains α and biases β: [72]

$$LN(\mathbf{z}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{(\mathbf{z} - \mu)}{\sigma} \odot \boldsymbol{\alpha} + \boldsymbol{\beta}, \qquad (3.5)$$

$$\mu = \frac{1}{D}\sum_{i=1}^{D} z_i, \quad \sigma = \sqrt{\frac{1}{D}\sum_{i=1}^{D}(z_i - \mu)^2}, \qquad (3.6)$$

Where, z_i is the i^{th} element of the vector z.

Decoder Layers:

Similar to the encoder, The decoder also consists of several layers of transformers, Each layer includes a masked multi-head self-attention mechanism to prevent the decoder from attending to future tokens. It also includes a multi-head cross-attention mechanism that attends to the encoder's output, each decoder layer has a feed-forward network, layer normalization, and residual connections.

Masked Multiple Self-Attention:

Similar to the encoder's self-attention but with masking to ensure that predictions for a position can depend only on the known outputs at earlier positions. This can be expressed in this formula [73]:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V \quad (3.7)$$

where M is a mask matrix that sets the scores of future tokens to negative infinity.

Encoder-Decoder Attention :

This allows the decoder to attend to all positions in the encoder's output. The formula is the same as the self-attention but uses the encoder's output as the key and value matrices:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (3.8)$$

[58]

where Q comes from the decoder and K and V come from the encoder.

Feed-Forward Neural Network: Similar to the encoder, the decoder also includes a feed-forward neural network applied at each position.

Layer Normalization: Applied similarly as in the encoder to the outputs of both the attention mechanisms and the feed-forward network.

The encoder converts an input sequence of symbol representations (x1... xn) into a sequence of continuous representations (z1, ..., zn). Using z, the decoder generates an output sequence (y1,...,ym) of symbols, producing one element at a time. The model operates autoregressively at each step, meaning it uses the previously generated elements to inform the next one.

Figure 3.3 shows a simplified schema of the BART encoder decoder architecture:

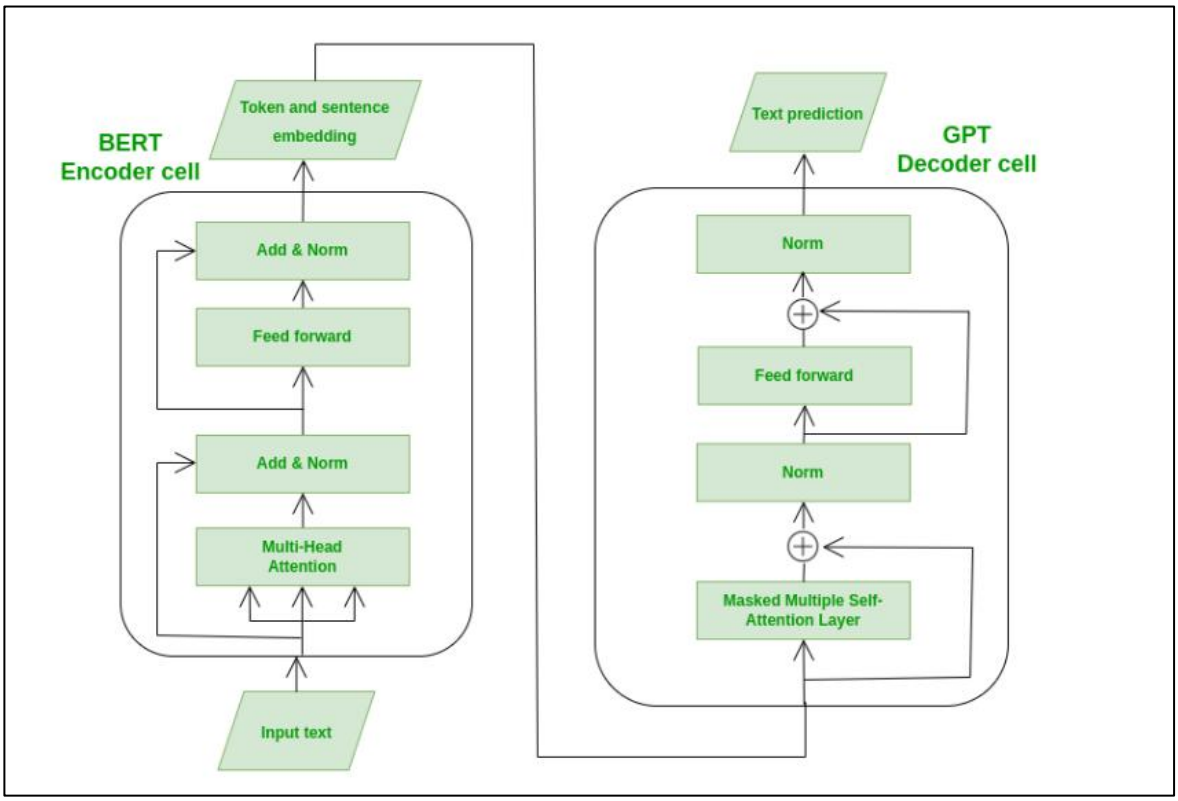


Figure3.3: BART encoder decoder architecture [74]

3.4 Model:

Recently, pre-trained models have emerged to address the issue of insufficient labeled data. These models are trained on vast amounts of unlabeled data to learn language representations. They can be fine-tuned for specific Natural Language Processing (NLP) tasks, often outperforming models built from scratch with limited labeled data.

The ability of pre-trained models to transfer learned knowledge to specific tasks has led to significant improvements across many NLP applications. Consequently, refining pre-trained models has become a standard and highly effective practice in the NLP field. This approach shows great promise for Arabic text summarization.

In this section, we present the pretrained model used for creation of our Arabic text summarization model, involving the partial refinement of the mBart deep learning model.

3.4.1 mBART-large-cc25:

After testing a few other models, including araBert, mBert , GPT-2, araT5 and mT5, we found that the mBART-large-cc25 model provided the best and most acceptable results for our needs. The mBART-large-cc25 model by Facebook on Hugging Face is a multilingual sequence-to-sequence mbart model pre-trained (not finetuned) for translation tasks. It uses a denoising autoencoder approach and is trained on 25 languages, making it effective for cross-lingual understanding and translation. The model can handle various language pairs, especially benefiting those with limited resources by leveraging shared multilingual representations. This helps improve translation quality across diverse languages.

The architecture of the pre-trained model follows a standard Seq2Seq Transformer architecture, with 12 layers of encoder and 12 layers of decoder, a model dimension of 1024 on 16 heads (~680M parameters). It includes an additional layer-normalization layer on top of both the encoder and decoder, which stabilizes training at FP16 precision. [75]

Beyond translation, mBART-large-cc25 can also be used for text summarization. Fine-tuning mBART on summarization datasets may create a powerful summarization model, Given an input document, it can generate a concise summary, capturing the essential information. The strengths of the model include its robust architecture, large capacity to capture complex linguistic patterns, and the ability to handle multiple languages effectively. These strengths make mBART-large-cc25 an optimal choice for high-quality text summarization.

3.4.2 Detailed Architecture of Fine-Tuning:

To create our Arabic text summarizer, we utilized and fine-tuned the pre-trained mBART-large-cc25 model. This approach allowed us to leverage the model's extensive multilingual capabilities and adapt it specifically for the task of summarizing Arabic text. Figure 3.4 illustrates the architecture of our fine-tuned model, detailing the various layers and processes involved in its construction.

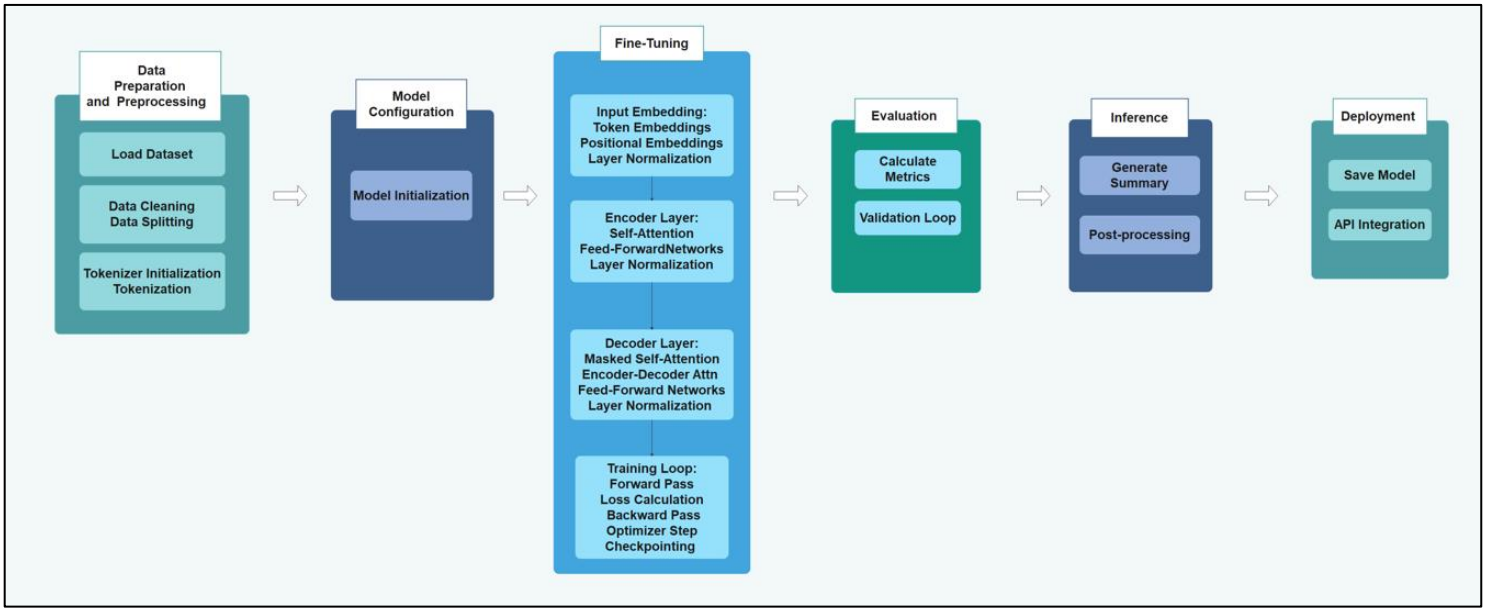


Figure 3.4: Our model architecture

A) Data Preparation and Preprocessing:

Data Loading: Load the XL-Sum dataset for Arabic.

Data Cleaning: Ensure data is cleaned and formatted correctly.

Data Splitting: Divide the dataset into training, validation, and test sets.

Tokenizer Initialization: Initialize MBartTokenizer with mbart-large-cc25.

Tokenization: Tokenize input texts and summaries.

B) Model Configuration:

Model Initialization: Initialize with pre-trained mbart-large-cc25 weights.

C) Fine-Tuning:

- **Input Embedding Layer:** Convert input tokens into embedding vectors.

Token Embeddings: Vector representations of tokens.

Positional Embeddings: Representations of token positions in the sequence.

Layer Normalization: Normalization of embeddings.

- **Encoder Layer:** Encode input sequences into internal representations.

Multi-Head Self-Attention: Attention mechanism allowing each token to focus on other tokens.

Feed-Forward Neural Networks: Neural networks to transform representations.

Residual Connections: Residual connections to stabilize learning.

Layer Normalization: Normalization after each sub-layer.

- **Decoder Layer:** Decode encoded representations into output sequences (summaries).

Masked Multi-Head Self-Attention: Masked attention mechanism for generating sequences.

Encoder-Decoder Attention: Attention mechanism to enable the decoder to focus on the encoded input.

Feed-Forward Neural Networks: Neural networks to transform representations.

Residual Connections: Residual connections to stabilize learning.

Layer Normalization: Normalization after each sub-layer.

- **Training Loop:** Refine the model on the training dataset, including forward and backward passes and optimizer steps:

Forward Pass: Compute model predictions.

Loss Calculation: Calculate loss by comparing predictions with actual targets.

Backward Pass: Backpropagate gradients to adjust weights.

Optimizer Step: Update model weights using the optimizer (e.g., AdamW).

Checkpointing: Periodically save the model to prevent data loss.

3.5 Conclusion:

In conclusion, this chapter has outlined the theoretical foundations and detailed steps involved in developing an Arabic text summarization model using mBART pretrained model. We discussed our proposed approach, the methodology, including dataset preparation, preprocessing, and the process of fine-tuning the model specifically for Arabic text summarization, taking into account the unique linguistic characteristics of the language.

In the next chapter, we will move from theory to practice, discussing the computational environment, programming frameworks, and libraries used in our implementation. We will also present and evaluate the results of our summarizer using ROUGE Metrics, comparing our findings with previous works in the field

.

CHAPTER IIV:

IMPLEMENTATION, RESULTS AND BENCHMARKING.

4.1 Introduction:

Following our comprehensive analysis of the system's architecture and an in-depth overview of the underlying models in the previous chapter, we now delve into the primary aspects of our study. This chapter will explore the computational environment, programming frameworks, and libraries employed in the implementation of our model. After that we will report the result of the proposed summarizer, finally, we will testing the performance of our model using ROUGE Metrics and compare the obtained results with two previous works in this field.

4.2 Implementation environment and tools:

A variety of technologies can be utilized to accomplish deep natural language processing tasks, many of which are available as open-source programs on the internet. This section details the numerous development frameworks and tools employed in the creation of this project.

4.2.1 Google Colaboratory:



Before introducing Google Colaboratory, we explore Jupyter Notebooks, the foundation of Colaboratory. Jupyter is an open-source, browser-based tool integrating interpreted languages, libraries, and visualization tools. Each notebook contains cells with script or markdown code, displaying outputs like text, tables, charts, and graphics. This setup simplifies sharing and replicating scientific work as experiments and results are self-contained. [76]

Google Colaboratory (or Colab) aims to make machine learning education and research more accessible, based on Jupyter, Colab notebooks function like Google Docs, allowing sharing and collaborative editing. They offer pre-configured Python runtimes with essential machine learning libraries like TensorFlow, Matplotlib, and Keras. Notebooks are preserved, but virtual machine data is lost after a period. However, files can be transferred to Google Drive. Colab also provides a GPU-accelerated runtime hosted on Google Cloud. [76]

Google Colab provides researchers with the capability to train complex deep learning models using various hardware accelerators, including CPUs, GPUs, and TPUs (Tensor Processing Units). [77]

4.2.2 Python:

Python is widely recognized as one of the most user-friendly and accessible programming languages. Its syntax is clear and straightforward, making the code easy to read and write. Python is also highly expressive, allowing developers to accomplish tasks with significantly fewer lines of code compared to languages like C++ or Java, it is a versatile cross-platform language with a comprehensive standard library and extensive third-party library support, enabling functionality across various operating systems without needing to build or compile code. [78]

This programming language is utilized in numerous application domains, including Data Science (as in our case), Web and Internet Development, Software Development, and Business Applications, among others.

In our project, we use Python version 3.10.12 as shown in **Figure 4.1:**

```
Python version
3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
Version info.
sys.version_info(major=3, minor=10, micro=12, releaselevel='final', serial=0)
```

Figure 4.1: Version of python used in this work.

4.2.3 PyTorch:

PyTorch is a highly optimized library that can manipulate data in the form of tensors. This library was developed by Facebook in 2016, it is used in creating deep learning models effectively using GPUs and CPUs [79], it will automatically select a GPU, if one is available, reverting to the CPU otherwise. [80]

It has an extensive choice of tools and libraries that supports on Computer Vision, Natural Language Processing (NLP) and many more ML programs. It allows developers to perform computations on Tensors with GPU acceleration and also helps in creating computational graphs. [81]

4.2.4 Transformers:



Transformers is a library dedicated to supporting Transformer-based architectures and facilitating the distribution and usage of a wide-variety of pretrained models in a centralized model hub. This hub supports users to compare different models with the same minimal API and to experiment with shared models on a variety of different tasks.

This library is an ongoing effort maintained by the team of engineers and researchers at HuggingFace with support from a vibrant community of over 400 external contributors.
Every model in the library is defined by three building blocks: (a) a tokenizer, which converts raw text to sparse index encodings, (b) a transformer, which transforms sparse indices to contextual embeddings, and (c) a head, which uses contextual embeddings to make a task-specific prediction. [82]

4.3 Implementation phase:

In this section we will explain the steps we took for build our models, also the various functions and parameters we used.

4.3.1 Loading and preprocessing the dataset:

The first step in our model is to import the dataset XL-sum, so we use the load_dataset function to import it from the Hugging Face datasets hub, than we select the 7000 first rows (we can not use all the data because of of resource limitations) as shown in **Figure 4.2**:

```
from datasets import load_dataset
# Load the dataset
dataset = load_dataset("csebuetnlp/xlsum", "arabic")

# Use the first 7000 rows
dataset = dataset['train'].select(range(7000))
```

Figure 4.2: load the dataset.

We convert the selected dataset into a pandas DataFrame, then we select only the necessary columns in our task, we select the column “text” which represents the original texts and “summary” which represents the summary, for easier manipulation and splitting into

training (80%), validation (10%), and test sets (10%) using sklearn library, this step is illustrated in **Figure 4.3**:

```
# Utiliser les premières 7000 lignes
dataset = dataset['train'].select(range(7000))
#Select relevant columns and convert to a pandas DataFrame
df = dataset.to_pandas()
df = df[['text', 'summary']]
# split into  train data (80%), validation data (10%) and test data (10%)
train_df, remaining_df = train_test_split(df, test_size=0.2, random_state=42)
val_df, test_df = train_test_split(remaining_df, test_size=0.5, random_state=42)
```

Figure 4.3: Selecting Relevant Columns and Splitting.

4.3.1.1 Most important functions used in the preprocessing step:

- **Remove duplicate and null rows:**

First, we remove duplicate and null rows using functions which are represented in **Figure 4.4**:

```
# Functions to remove null rows and duplicate rows
def remove_null_rows(df):
    return df.dropna()

def remove_duplicate_rows(df):
    return df.drop_duplicates()

# Apply functions to remove null and duplicate rows
df = remove_null_rows(df)
df = remove_duplicate_rows(df)
```

Figure 4.4: the functions used to remove null and duplicate rows.

- **Text normalization:**

We remove diacritics in both Text and Summary column, using the function remove_diacritics which is shown in Figure 4.5:

```
def remove_diacritics(input_text):
    vowelization = re.compile(r""" ّ|ً|ٌ|ٍ|َ|ُ|ِ|ْ|ـ""", re.VERBOSE)
    out_text = re.sub(vowelization, '', input_text)
    return out_text
```

Figure 4.5: the function used to remove diacritics.

After that we normalize Arabic letters to handle variations and simplify representation, we employed the function replace_letters which is shown in Figure 4.6:

```
def replace_letters(input_text):
    replace = {"ٳ": "ا", "ٵ": "ء", "ٰ": "ا", "ٙ": "ا"}
    replace = dict((re.escape(k), v) for k, v in replace.items())
    pattern = re.compile("|".join(replace.keys()))
    out_text = pattern.sub(lambda m: replace[re.escape(m.group(0))], input_text)
    return out_text
```

Figure 4.6: the function used to normalize Arabic letters.

- **Text cleaning:**

We use diffrent functions for cleaning and preparing our data, we use the function remove_punctuation to delete the Arabic and English punctuations, and we remove unnicessary characters like links using the function delete_links which is represented in **Figure 4.7**:

```
import string
# define the functions of cleaning and preprocessing
def delete_links(input_text):
    pattern = r'''(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)
    (?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]+|(\([^\s()<>]+\)))*\)|
    [^\s`!()\[\]{};:'".,<>?«»“”‘’]))'''
    out_text = re.sub(pattern, ' ', input_text)
    return out_text
# Define Arabic punctuation characters
arabic_punctuation = '''_«»؟؛.“،~_،:"'،؛١٢٣٤٥٦٧٨٩'''
# Get English punctuation characters
english_punctuation = string.punctuation
# Combine both lists into a single list
all_punctuation = arabic_punctuation + english_punctuation
def remove_punctuation(input_text):
    # Create translation table to remove punctuations
    translator = str.maketrans('', '', all_punctuation)
    # Remove punctuation using translation table
    cleaned_text = input_text.translate(translator)
    return cleaned_text
```

Figure 4.7: The functions of text cleaning.

- **Tokinization:**

We tokenize the preprocessed data to prepare it for input into the summarization model; we use the tokenizer of our chosen model mbart-large-cc25. Firstly, we import it from hugging face hub as shown in **Figure 4.8**.

```
from transformers import AutoTokenizer
# load the mbart tokenizer
tokenizer = AutoTokenizer.from_pretrained("facebook/mbart-large-cc25")
```

Figure 4.8: Load the associated pre-trained tokenizer of the model.

Secondly, we build a function that preprocesses our data by applying the previously mentioned preprocessing functions to both the text and summary rows, as well as passing them to the tokenizer with the `truncation=True` option. This ensures that inputs longer than the model's maximum length of 512 tokens will be truncated. Similarly, for the labels, which have a maximum length of 128 tokens, truncation is applied to ensure they fit within this limit. We pad instances to the longest length in the batch rather than across the entire dataset because padding will be handled later, typically in a data collator function. **Figure 4.9** represents this function.

```
def preprocess_function(examples):
    inputs = examples['text']
    targets = examples['summary']

    for i in range(len(inputs)):
        inputs[i] = delete_links(inputs[i])
        inputs[i] = replace_letters(inputs[i])
        inputs[i] = remove_diacritics(inputs[i])
        inputs[i] = remove_punctuation(inputs[i])
    for i in range(len(targets)):
        targets[i] = delete_links(targets[i])
        targets[i] = replace_letters(targets[i])
        targets[i] = remove_diacritics(targets[i])
        targets[i] = remove_punctuation(targets[i])

    model_inputs = tokenizer(inputs, max_length=512, truncation=True, padding="max_length")
    labels = tokenizer(targets, max_length=128, truncation=True, padding="max_length")
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

# Apply preprocessing to the training, validation, and test datasets
train_dataset = train_dataset.map(preprocess_function, batched=True)
val_dataset = val_dataset.map(preprocess_function, batched=True)
test_dataset = test_dataset.map(preprocess_function, batched=True)
```

Figure 4.9: the preprocess function used.

The data has been preprocessed and is now ready to be fed to the chosen model.

4.3.2 Fine-tuning the model:

We want to fine-tune our chosen model, mBART-large-cc25, on our data, so we import this model from Hugging Face, as shown in **Figure 4.10**.

```
from transformers import AutoModelForSeq2SeqLM
# load the mbart model
model = AutoModelForSeq2SeqLM.from_pretrained("facebook/mbart-large-cc25")
```

Figure 4.10: Load the chosen model.

4.3.2.1 Trainer API:

We will fine-tune the model using the Trainer, which is a class in PyTorch provides a comprehensive API for standard training scenarios. This API supports distributed training on multiple GPUs/TPUs, mixed-precision training with NVIDIA Apex, and PyTorch Native .
The Trainer requires the following three components:

- The Seq2SeqTrainingArguments which contain the arguments using in the training and the optimizer, which are :
  a. **Learning Rate:** The learning rate is a hyper parameter that controls the adjustment of the neural network's weights in response to the loss gradient. It defines how quickly the neural network updates the concepts it has learned.
  b. **Optimization Function:** The optimization function, or optimizer, is a method or algorithm used to adjust the weights of the neural network to minimize the loss function.
  c. **Batch Size:** The batch size is a hyper parameter that determines the number of samples processed before the model's internal parameters are updated.
  d. **Number of Epochs:** The number of epochs is a hyper parameter that defines the number of complete passes through the training dataset.
  e. **Gradient Accumulation Steps:** Gradient accumulation steps is a technique where gradients are accumulated over multiple batches before performing a weight update. It effectively allows for a larger batch size without requiring additional memory.

The **table 4.1** illustrates our chosen arguments:

Train batch size	Evaluation batch size	Gradient accumulation steps	Number of Epochs	Learning Rate	Optimizer function
2	2	2	3	5e-5	AdamW

Table 4.1: The Seq2SeqTrainingArguments used.

To determine the optimal learning rate, we conducted multiple training sessions using various values, ultimately achieving the best result with a learning rate of 5e-5, we use batch size= 2 in the train and evaluation Because of resource limitations, we were unable to try any

additional higher values. And also because of the long time of training and the limited Time allowed for using google colab Environment we use only 3 epochs.

```
# define train arguments
training_args = Seq2SeqTrainingArguments(
    output_dir='./results',
    evaluation_strategy="epoch",
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    gradient_accumulation_steps=2,
    weight_decay=0.01,
    learning_rate=5e-5,
    save_total_limit=2,
    num_train_epochs=3,
    predict_with_generate=True,
    logging_dir='./logs',
    logging_steps=10,
    fp16=True,
)

# define  AdamW optimizer
optimizer = AdamW(
    model.parameters(),
    lr=training_args.learning_rate,
    weight_decay=training_args.weight_decay,
)
```

Figure 4.11: Defining training arguments.

- The DataCollatorForSeq2Seq which is a special type of data collator; it takes a list of dataset elements as input and create batches of data. These elements are of the same type as those in the training and evaluation datasets. Data collators can perform various processing tasks to create batches, such as padding. In our case, DataCollatorForSeq2Seq will dynamically pad both the inputs and labels to the maximum length within each batch.

We define this data collator as shown in **Figure 4.12**.

```
from transformers import DataCollatorForSeq2Seq
# create  DataCollator for the sequence generation
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
```

Figure 4.12: Defining the data collator.

- The function which is used for evaluate our model during the training, this function uses the validation dataset for evaluation after every set of predictions as well as help the model tune its hyper parameters.

We use ROUGE-N (ROUGE1, ROUGE2) which Focuses on the uni-gram similarity between the ATS summary and the reference summary. It employs an N-gram recall algorithm to compare the system and reference summaries, the formula that calculates ROUGE-N is [24]:

$$\text{ROUGE-}n = \frac{\sum_{C \in RSS} \sum_{gram_n \in C} \text{Count}_{match}(gram_n)}{\sum_{C \in RSS} \sum_{gram_n \in C} \text{Count}(gram_n)}, \quad (4.1)$$

Where $\text{Count}_{match}(\text{gram}_n)$ is the maximum number of n-grams co-occurring in a candidate summary and a reference summary and $\text{Count}(\text{gram}_n)$ is the number of n-grams in the reference summary. [24]

Also we use ROUGE-L which based on the longest common sub-sequences (LCS) between human-generated and automatically generated summaries. It evaluates the ratio of the size of the LCS of two summaries to the size of the reference summary. The formula that calculates ROUGE-L is [83]:

$$P_{lcs} = \frac{LCS(X,Y)}{n} \quad (4.2)$$

$$R_{lcs} = \frac{LCS(X,Y)}{m} \quad (4.3)$$

$$F_{lcs} = \frac{(1+\beta^2) R_{lcs} P_{lcs}}{R_{lcs} + \beta^2 P_{lcs}} \quad (4.4)$$

Where:

- X: is the reference summary sentence.
- Y: is the candidate summary sentence.
- LCS(X,Y): is the length of a longest common subsequence of X and Y.
- m: is the length of reference summary sentence X.
- n: is the length of candidate summary sentence Y.
- ß: is set to a very big number, it is the relative importance of PLCS and RLCS which equals:

$$\beta = \frac{P_{lcs}}{R_{lcs}} \tag{4.5}$$

We also use ROUGE-LSUM, which works very similarly to ROUGE-L. ROUGE-LSUM is calculated using matching union Longest Common Subsequence (LCS) sequences between a candidate summary and a reference summary; it is calculated using the following formula [84]:

$$R_{lcssum} = \frac{\sum_{i=1}^{u} LCS_{\cup}(r_{i,S})}{m} \tag{4.6}$$

$$P_{lcssum} = \frac{\sum_{i=1}^{u} LCS_{\cup}(r_{i,S})}{n} \tag{4.7}$$

$$F_{lcssum} = \frac{(1+\beta^2)R_{lcs}P_{lcs}}{R_{lcs}+\beta^2 P_{lcs}} \tag{4.8}$$

Where a candidate summary *sj* that has v number of sentences and a total of n words, and a reference summary *rj* that has u number of sentences and a total of m words. [84]

We define compute_metrics function for calculate all this ROUGE metrics during the training as shown in **Figure 4.13** and we print only the ROUGE F-measure at the end of each epoch.

```
#import the load metrics
from datasets import load_metric
# load ROUGE metrics
rouge = load_metric("rouge", experiment_id="main",trust_remote_code=True)
# Define compute_metrics function
def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)
    labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)

    result = rouge.compute(predictions=decoded_preds, references=decoded_labels, use_stemmer=True)
    result = {key: value.mid.fmeasure * 100 for key, value in result.items()}

    prediction_lens = [np.count_nonzero(pred != tokenizer.pad_token_id) for pred in predictions]
    result["gen_len"] = np.mean(prediction_lens)

    return {k: round(v, 4) for k, v in result.items()}
```

Figure 4.13: the compute_metrics function.

Finally, we must give all of this information, as well as our datasets and the metric, to the Seq2SeqTrainer. And start the training as shown in **Figure 4.14**:

```
# Define the trainer
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    optimizers=(optimizer, None),  # Use AdamW optimizer
)

# train the model
trainer.train()
```

Figure 4.14: Defining the Trainer and strat training.

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

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
1	1.371900	1.388338	1.071400	0.000000	1.095200	1.095200	100.801400
2	1.129100	1.327461	2.095200	0.142900	2.095200	2.071400	67.497100
3	0.894700	1.356652	1.381000	0.071400	1.381000	1.404800	77.525700

Figure 4.15: Measurements of 3 epochs.

4.3.3 Prediction:

The Trainer provides a method for testing the model on a test dataset termed Predict. It generates predictions on a test set, so we use it and makes the prediction as shown in **Figure 4.16.**

```
# generate the summaries with the model on test data
predictions = trainer.predict(test_dataset)
```

Figure 4.16: Apply the prediction function on the test dataset.

4.4 Evaluation of model:

In this section, we are going to evaluate the summaries resulting from our text summarizer 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, the **Figure 4.17** represents this step and the result obtained.

```
from datasets import load_metric
# load ROUGE metric
rouge_metric = load_metric("rouge")

# generate the summaries with the model on test data
predictions = trainer.predict(test_dataset)

# Calculate the ROUGE scores
rouge_results = rouge_metric.compute(predictions=predictions.predictions, references=predictions.label_ids)

# print the rouge scores
print("Rouge1",rouge_results['rouge1'].mid.fmeasure*100)
print("Rouge2",rouge_results['rouge2'].mid.fmeasure*100)
print("RougeL",rouge_results['rougeL'].mid.fmeasure*100)
```

```
Rouge1 52.59523809523814
Rouge2 41.257142857142945
RougeL 52.59523809523811
```

Figure 4.17: Evaluation of our model using ROUGE metric.

These results are indeed highly satisfactory and commendable. The ROUGE-1 score of 52.59 indicates that our model effectively captures the keywords from the source text in its generated summary. It's a good result, suggesting adequate understanding of the text content. Additionally, ROUGE-2 = 41.25 shows that our model is capable of grasping some relationships between consecutive word pairs in the source text, while ROUGE-L = 52.59 indicates that our model aligns well with whole sentences or sequences of words in the reference summary. This is encouraging as it shows consistency in how our model generates summaries. Our model demonstrates a robust capability to extract crucial information from source texts and produce coherent summaries. These performance metrics stand favorably when compared to those reported in previous research.

Benchmarking:

Now, we will compare our results with the results of two previous works that are referenced in the related work section. The table 4.2 illustrates these results with our results.

Ranking	Model	ROUGE-1	ROUGE-2	ROUGE-L
1	T5-arabic-base [33]	62.84	54.84	61.98
2	mBERT Eng [32]	60.32	25.79	56.91
3	**mBART-cc25(Our model)**	**52.595**	**41.257**	**52.595**
4	mBERT Hun [32]	47.02	19.72	39.29
5	mBART Rus [32]	32.1	14.2	25.7

Table 4.2: A comparison of our model with previous works.

When compared to previous research, our model ranks 3rd out of 5, which further highlights its effectiveness and reliability in the field of text summarization. This comparison is logical given the resources and time available for this work. Our results are commendable, especially considering the constraints. With more resources, better hardware, and a larger, more diverse dataset, our model's performance could have been even better.

4.5 Development interface:

We have implemented an interface using Flask, one of the micro-frameworks used to create web applications in Python [85] to facilitate the use of our model for text summarization.

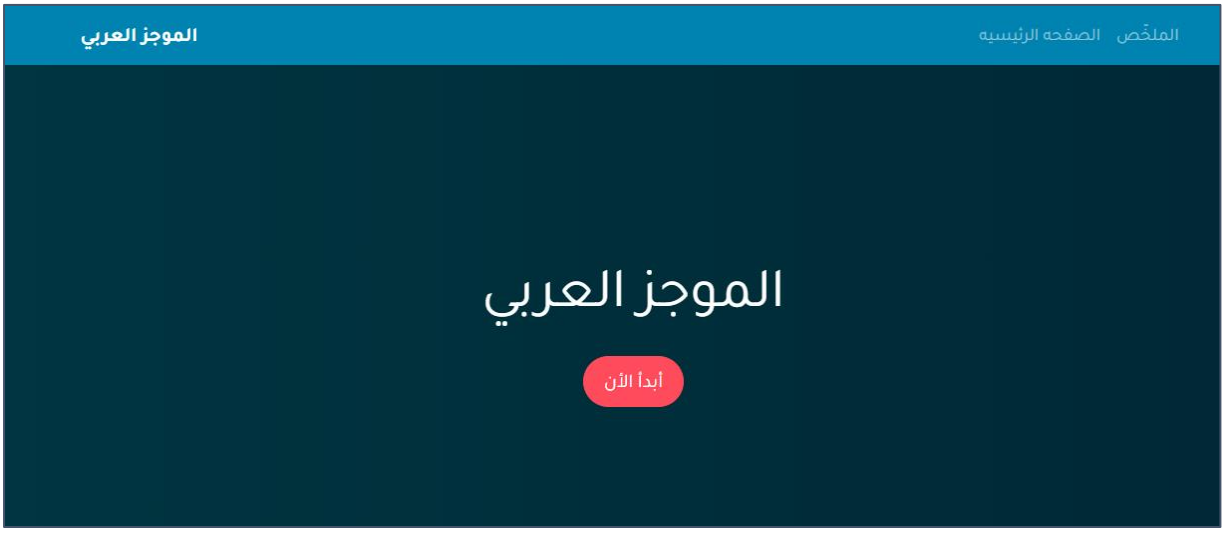


Figure 4.18: The main page of our interface.

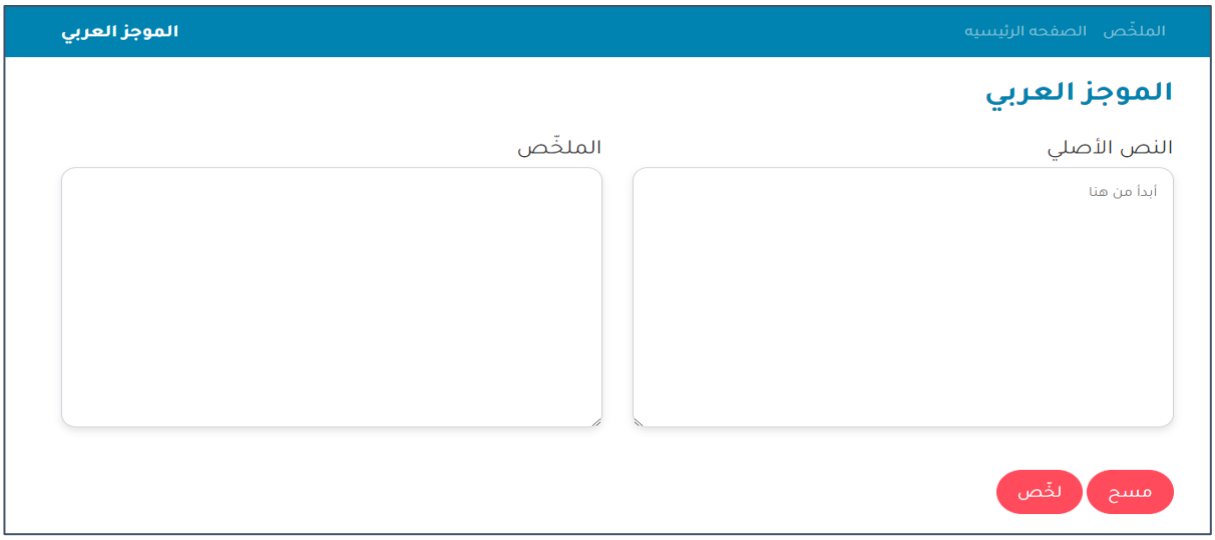


Figure 4.19: The automatic summarization page of our model.

Figure 4.19 represents the automatic summarization page of our model which consists of:

- Two text fields: The first field is for entering the text to be summarized, and the second field is for displaying the generated summary.
- A clear button "مسح" that helps the user clear the text from the fields.
- A submit button "لخص" that, when clicked, triggers the automatic summary generation process.

This page aims to provide an easy-to-use interface for users to input their text and receive a concise summary of it.

The **Figure 4.20** illustrates an application of our model, which automatically generates a summary of text extracted from the web.



Figure 4.20: Application of our model on text from the web using our interface.

4.6 Conclusion:

In this chapter, we first discussed our Implementation environment, tools and functions utilized throughout our project; we also highlighted our results which demonstrated the effectiveness of our model, outperforming many state-of-the-art solutions in Arabic text summarization.

Developing an effective Arabic text summarization model requires addressing the unique linguistic challenges posed by the language. By leveraging the advanced capabilities of mBART, implementing meticulous preprocessing, and fine-tuning steps, we have laid a solid foundation for our summarization model.

General Conclusion and future work:

This study is useful in understanding one of the most prominent tasks within the scope of natural language processing, particularly in the Arabic language, which is identifying the latest methods used to generate summaries that accurately capture the main ideas of a text. The task of automatic summarization has garnered significant attention in recent years, particularly with the advent of advanced techniques that have greatly improved the ability to create summaries that are similar to those crafted by humans, specifically in the Arabic language.

In this study, we have investigated various pre-trained models and introduced a novel approach for summarizing Arabic text based on the MBART model. We designed, developed, and compared the results of our proposed summarizer. In general, our summarizer yields comparable and satisfactory outcomes relative to other work in this domain on the Arabic language, despite the challenges posed by the Arabic language in this type of work. However, due to hardware limitations and the unavailability of the necessary environment, we were unable to train our model for extended periods or test it on other datasets to determine its effectiveness.

In the future, we aim to enhance our model's performance by incorporating new datasets that span various domains. This expansion will help our model to generate more coherent summaries tailored to specific fields. Additionally, we plan to refine our training process through advanced fine-tuning techniques, optimizing its ability to handle nuanced nuances in different types of content. Given the potential versatility of our model, future experiments will explore its application in diverse domains such as scientific papers, social media content, and legal documents, across different languages like German, French, and Spanish, further demonstrating its robustness and adaptability.

Bibliography

[1] K. Kumar and G. S. M. Thakur, "Advanced applications of neural networks and artificial intelligence: A review," *International journal of information technology and computer science,* pp. 57 - 68, 2012.

[2] S. Mysore, "Role Of Artificial Intelligence In Grid Modernization: Exploring How Ai Can Enhance Grid Management, Predict Energy Demand, And Optimize Renewable Energy Usage," *International Research Journal of Modernization in Engineering Technology and Science ,* vol. 6, no. 1, 2024.

[3] M. R. Turchioe, A. Volodarskiy, J. Pathak, D. N. Wright, . J. E. Tcheng and D. Slotwiner, "Systematic review of current natural language processing methods and applications in cardiology," *Heart,* vol. 108, pp. 909-916, 2022.

[4] . A. Koli, D. Khurana, . K. Khatter and S. Singh, "Natural language processing: state of the art, current trends and challenges," *arXiv preprint,* vol. 82, pp. 3,7, 2017.

[5] J. Yin, T.-T. Goh, . B. Yang and Y. Xiaobin, "Conversation technology with micro-learning: the impact of chatbot-based learning on students' learning motivation and performance.," *J. Educ. Comput. Research.,* vol. 59, pp. 154,177, 2021.

[6] Z. Wang, "Modern Question Answering Datasets and Benchmarks: A Survey," *arXiv preprint arXiv:2206.15030,* 2022.

[7] G. Vinodhini and R. M. Chandrasekaran, "Sentiment Analysis and Opinion Mining: A Survey," *International Journal,* vol. 2, no. 6, 2012.

[8] J. Lin, "Summarization," *Springer, New York, NY,* 2018.

[9] Ö. Sahin, "Text Summarization," in *Develop Intelligent iOS Apps with Swift Apress*, Berkeley,

2021.

[10] R. Z. Al-Abdallah and A. T. Al-Taani, "Arabic Text Summarization using Firefly Algorithm," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, Dubai, 2019, pp. 61-65.

[11] T. Jo, "Text Summarization," *Text Mining. Studies in Big Data,* vol. 45, 2019.

[12] K. Shaalan, S. Siddiqui, M. Alkhatib and A. Abdel Monem, "Challenges in Arabic natural language processing.," *Computational linguistics, speech and image processing for arabic language,* pp. 59-83, 2019.

[13] N. Y. Habash, "Introduction to Arabic natural language processing.," *Morgan & Claypool Publishers,* 2010.

[14] D. Alahmadi, A. Wali and S. Alzahrani, "TAAM: Topic-aware abstractive arabic text summarisation using deep recurrent neural networks," *Journal of King Saud University - Computer and Information Sciences,* vol. 34, no. 6, pp. 2651-2665, 2022.

[15] A. Farghaly and K. Shaalan, "Arabic natural language processing: Challenges and solutions," *Farghaly, Ali, and Khaled Shaalan. "Arabic natural language processing: ChACM Transactions on Asian Language Information Processing (TALIP),* vol. 8, no. 4, pp. 1-22, 2009.

[16] M. A. Aabed, S. M. Awaideh, A. R. M. Elshafei and A. A. Gutub, "Arabic diacritics based steganography," *IEEE International Conference on Signal Processing and Communications,* pp. 756-759, 2007.

[17] S. Lagrini, M. Redjimi and N. Azizi, "A Survey of Extractive Arabic Text Summarization Approaches," in *Arabic Language Processing: From Theory to Practice.*, vol. 782, 2018, p. 159–171.

[18] W. S. El-Kassas, C. R. Salama, A. A. Rafea and H. K. Mohamed, "Automatic text summarization: A comprehensive survey," *Expert Systems with Applications,* vol. 165, 2021.

[19] D. Yadav, J. Desai and A. K. Yadav, "Automatic Text Summarization Methods: A Comprehensive Review," *arXiv preprint arXiv:2204.01849,* 2022.

[20] G. Sharma and . D. Sharma, "Automatic Text Summarization Methods: A Comprehensive Review," *SN COMPUT. SCI.,* vol. 4, 2023.

[21] B. Khan, Z. A. Shah, M. Usman, I. Khan and B. Niazi, "Exploring the Landscape of Automatic Text Summarization: A Comprehensive Survey," *IEEE Access,* vol. 11, 2023.

[22] A. Elsaid, A. Mohammed, L. F. Ibrahim and M. M. Sakre, "A Comprehensive Review of Arabic Text Summarization," *IEEE Access,* vol. 10, pp. 38012-38030, 2022.

[23] M. F. Mridha, A. A. Lima, K. Nur, S. C. Das, M. Hasan and M. M. Kabir, "A Survey of Automatic Text Summarization: Progress, Process and Challenges," *IEEE Access,* vol. 9, pp. 156043-156070, 2021.

[24] J. Steinberger and K. Jezek, "Evaluation measures for text summarization," *Computing and Informatics ,* vol. 28, no. 2, p. 251, 2009.

[25] S. Moulay Lakhdar and M. Chéragui, "Building an Extractive Arabic Text Summarization Using a Hybrid Approach," in *Smaïli, K. (eds) Arabic Language Processing: From Theory to Practice. ICALP 2019*, vol. 1108, Communications in Computer and Information Science, 2019.

[26] M. Bani-Almarjeh and M.-B. Kurdy, "Arabic abstractive text summarization using RNN-based and transformer-based architectures," in *Information Processing & Management*, vol. 60, 2023.

[27] D. Suleiman and A. Awajan, "Deep learning based abstractive arabic text summarization using two layers encoder and one layer decoder," *Theoretical and Applied Information Technology,* pp. 3233-3244, 2020.

[28] Y. Wazery, M. Saleh, A. Alharbi and A. Ali, "Abstractive Arabic text summarization based on deep learning," *Computational Intelligence and Neuroscience,* 2022.

[29] A. M. Rush, S. Chopra and J. Weston, "A neural attention model for abstractive sentence summarization," *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing,* pp. 379-389, 2015.

[30] R. Nallapati, B. Zhou, C. Gulcehre and B. Xiang, "Abstractive text summarization using sequence-to-sequence RNNs and beyond.," *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning,* pp. 280-290, 2016.

[31] P. Kouris, G. Alexandridis and A. Stafylopatis, "Abstractive text summarization based on deep learning and semantic content generalization," *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics,* p. 5082–5092, 2019.

[32] M. Kahla, Z. G. Yang and N. Attila, "Cross-lingual fine-tuning for abstractive Arabic text summarization," *Proceedings of the international conference on recent advances in natural language processing (ranlp 2021),* pp. 655-663, 2021.

[33] K. Alissa, I. Qusai and R. M. Duwairi, "Arabic News Summarization based on T5 Transformer Approach," *14th International Conference on Information and Communication Systems (ICICS),* pp. 1-7, 2023.

[34] W. AlSanie, G. Kotsis, D. Taniar, S. Bressan, I. Ibrahim and S. Mokhtar, "Towards a suitable rhetorical representation for Arabic text summarization," in *iiWAS*, vol. 196, Austrian Computer Society, 2005, p. 535–542.

[35] F. Douzidia and G. L. Lapalme, "an Arabic summarization system," in *Proceedings of 2004 Document Understanding Conference (DUC 2004)*, Boston, 2004, p. 128–135.

[36] Q. Al-Radaideh and D. Bataineh, "A hybrid approach for Arabic text summarization using domain knowledge and genetic algorithms," in *Cogn. Comput.*, vol. 10, 2018, p. 651–669.

[37] A. Ibrahim, T. Elghazaly and M. Gheith, "A novel arabic text summarization model based on rhetorical structure theory and vector space model," in *Int J Comput Linguist Nat Lang Proces*, vol. 2, 2013, pp. 480-485.

[38] I. Sobh, N. Darwish and M. Fayek, "An optimized dual classification system for Arabic extractive generic text summarization," in *In the 7th Conference on Language Engineering*, 2006, p. 149–154.

[39] N. Alami, M. Meknassi, S. A. Ouatik and N. Ennahnahi, "Arabic text summarization based on graph theory," in *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, 2015, pp. 1-8.

[40] R. Elbarougy, G. Behery and A. E. Khatib, "Extractive arabic text summarization using modified pagerank algorithm," *Egyptian Informatics Journal,* vol. 21, no. 2, pp. 73-81, 2020.

[41] A.D.Dongare, R.R.Kharde and A. D.Kachare, "Introduction to Artificial Neural Network," *International Journal of Engineering and Innovative Technology (IJEIT),* vol. 2, no. 1, 2012.

[42] H. Kukreja, B. N, S. C. S and K. S, "AN INTRODUCTION TO ARTIFICIAL NEURAL NETWORK," *Int J Adv Res Innov Ideas Educ,* pp. 27-30.

[43] K. S and D. A. G. S, "Neural Network Approach for Processing Substation Alarms," *International Journals of Power Electronics Controllers and Converters.*

[44] Y. Bengio, "Learning deep architectures for AI," *Now Publishers Inc,* 2009.

[45] W. Z. e. al, "On definition of deep learning," *2018 World automation congress (WAC). IEEE,* p. 234, 2018.

[46] R. Shyam, "Convolutional Neural Network and its Architectures," *Journal of Computer Technology & Applications,* vol. 12, no. 2, p. 6–14, 2021.

[47] T. Katte, "Recurrent neural network and its various architecture types.," *International Journal of Research and Scientific Innovation 5.3,* pp. 124-129, 2018.

[48] H. Salehinejad, S. Sankar, J. Barfett, E. Colak and S. Valaee, "Recent Advances in Recurrent Neural Networks," *arXiv:1801.01078,* 2018.

[49] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing,* vol. 45, no. 11, pp. 2673-2681, 1997.

[50] S. Zargar, "Introduction to sequence learning models: RNN, LSTM, GRU," *Department of Mechanical and Aerospace Engineering, North Carolina State University,* 2021.

[51] H. Sak, W. Andrew, S. W and B. Françoise, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," 2014.

[52] K. Cho, B. Van Merriënboer, D. Bahdanau and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259,* 2014.

[53] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555,* 2014.

[54] I. Sutskever, O. Vinyals and Q. V. Le., "Sequence to sequence learning with neural networks," *Advances in neural information processing systems 27,* 2014.

[55] S. Kostadinov, "Understanding Encoder-Decoder Sequence to Sequence Model".

[56] T. Shi, Y. Keneshloo, N. Ramakrishnan and C. K. Reddy, "Neural Abstractive Text Summarization with Sequence-to-Sequence Models," *ACM Transactions on Data Science,* vol. 2, no. 1, pp. 1-37, 2021.

[57] D. Suleiman and A. Awajan, "Deep Learning Based Abstractive Text Summarization: Approaches, Datasets, Evaluation Measures, and Challenges," 24 August 2020.

[58] Vaswani, Ashish, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30*, 2017, p. 5998–6008.

[59] P. Dufter, M. Schmitt and H. Schütze, "Position Information in Transformers: An Overview," *Computational Linguistics,* vol. 48, no. 3, p. 733–763, 2022.

[60] T. Lin, Y. Wang, X. Liu and X. Qiu, "A survey of transformers," *AI Open,* pp. 111-132, 2022.

[61] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805,* 2018.

[62] B. Ghojogh and A. Ghodsi, "Attention mechanism,transformers, BERT, and GPT: tutorial and survey," 2020.

[63] F. A. Acheampong, H. Nunoo-Mensah and W. Chen, "Transformer Models for Text-based Emotion Detection: A Review of BERT-based Approaches," *The Artificial Intelligence Review 54.8,* pp. 5789-5829, 2021.

[64] R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova and J. Lin, "Distilling task-specific knowledge from bert into simple neural networks," *arXiv,* p. 8, 2019.

[65] H. El Moubtahij, H. Abdelali and E. B. Tazi, "AraBERT transformer model for Arabic comments and reviews analysis," *IAES International Journal of Artificial Intelligence,* vol. 11, no. 1, pp. 379-387, 2022.

[66] G. Y. e. al., "GPT (Generative Pre-trained Transformer) – A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions," *IEEE Access.*

[67] M. e. a. LEWIS, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.," *arXiv preprint arXiv:1910.13461,* 2019.

[68] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li and P. J. Liu , "Exploring the Limits of Transfer Learning with a Unified," *Journal of Machine Learning Research,* vol. 21, no. 140, pp. 1-67, 2020.

[69] P. Pęzik, A. Mikołajczyk, A. Wawrzyński, B. Nitoń and . M. Ogrodniczuk, "Keyword Extraction from Short Texts with a Text-to-Text Transfer Transformer," Singapore, 2022.

[70] T. Hasan, . A. Bhattacharjee, M. S. Islam, K. Samin, Y.-F. Li, Y.-B. Kang, M. S. Rahman and R. Shahriyar, "XL-Sum: Large-Scale Multilingual Abstractive Summarization for 44," *arXiv preprint arXiv:2106.13822,* pp. 4693-4703, 2021.

[71] H. Chouikhi, H. Chniter and F. Jarray, "Arabic Sentiment Analysis Using BERT Model," 2021.

[72] J. L. Ba, J. R. Kiros and G. E. Hinton, "Layer Normalization," *arXiv preprint,* 2016.

[73] J. Alammar, "The Illustrated Transformer," 2018. [Online]. Available: https://jalammar.github.io/illustrated-transformer/. [Accessed 21 06 2024].

[74] G. "BART Model for Text Auto Completion in NLP," 2023. [Online]. Available: https://www.geeksforgeeks.org/bart-model-for-text-auto-completion-in-nlp/. [Accessed 21 06 2024].

[75] Y. Liu, J. Gu, N. Goyal, . X. Li, S. Edunov, M. Ghazvininejad, M. Lewis and L. Zettlemoyer, "Multilingual Denoising Pre-training for Neural Machine Translation," *Transactions of the Association for Computational Linguistics,* vol. 8, pp. 726-742, 2020.

[76] T. Carneiro, R. V. M. D. NóBrega, T. Nepomuceno, G. -B. Bian, V. H. C. D. Albuquerque and P. P. R. Filho, "Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications," *IEEE Access,* vol. 6, pp. 61677-61685, 2018.

[77] V. Sharma, G. K. Gupta and M. Gupta, "Performance Benchmarking of GPU and TPU on Google Colaboratory for Convolutional Neural Network.," *Applications of Artificial Intelligence in Engineering,* p. 639, 2021.

[78] M. Summerfield, "Programming in Python 3: A Complete Introduction to the Python Language ", 2008, pp. 1-393.

[79] T. M. Arif, "Introduction to PyTorch," *Introduction to Deep Learning for Engineers: Using Python and Google Cloud Platform. Cham: Springer International Publishing,* pp. 11-15, 2020.

[80] I. Vasilev, D. Slater, G. Spacagna, P. Roelants and V. Zocca, "Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow," *Packt Publishing Ltd,* 2019.

[81] A. S. Saabith, T. Vinothraj and M. Fareez., "Popular python libraries and their application domains," *International Journal of Advance Engineering and Research Development,* vol. 7, no. 11, 2020.

[82] T. e. a. Wolf, "Huggingface's transformers: State-of-the-art natural language processing.," *arXiv preprint arXiv:1910.03771,* 2019.

[83] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," *Text summarization branches out,* pp. 74-81, 2004.

[84] E. Andersson, "Methods for increasing cohesion in automatically extracted summaries of Swedish news articles: Using and extending multilingual sentence transformers in the data-processing stage of training BERT models for extractive text summarization.," p. 37, 2022.

[85] N. Idris, C. F. Mohd Foozy and P. Shamala, "A Generic Review of Web Technology: DJango and Flask," *International Journal of Advanced Science Computing and Engineering,* vol. 2, no. 1, p. 34–40, 2021.

[86] GeeksforGeeks, "BART Model for Text Auto Completion in NLP," [Online]. Available: https://www.geeksforgeeks.org/bart-model-for-text-auto-completion-in-nlp/. [Accessed 20 06 2024].