People's Democratic Republic of Algeria Ministry of Higher Education and Scientific Research

SaaD Dahleb University Blida -1-

Science Faculty

Computer Science Departement

Master 2 Thesis

in Computer Science

Branch : Natural Language Processing

Entitled :

Multi-Documents Summarizing Using Transformers Neural Networks

Realised By :

KHELIFA Souhail.

ZOUBIR Chouaib Wali-Eddine.

Promoted By :

Mr.KAMECHE Abdellah.

Presented in front of :

Mme.Abed.

Mme.BERRAMDANE.

Date : ... /... /....

Acknowledgement :

First of all, I would like to thank My promoter Mr.KAMECHE who gave me all the help I needed towards this project And to my partner KHELIFA Souhail.

I would like to thank the two pearls that stood beside me during not only my project but all my life, To my brother who taught me everything I needed and to my father who gave me the wisdom required to move forward, I would be forever in their debts.

To All my Friends who were there for me in times of need : My TS-Squad, my Labbo-Buddies and My Council ! And to all ClubS Members ! You guys made my life more awesome.

-ZOUBIR Chouaib Wali-Eddine.

_____-

First of all, I would like to thank Allah for all the strength he gave me.

I would like to thank my promoter Mr.Kameche and my partner ZOUBIR Chouaib for the experience and the help they gave me.

I would like to thank my parents who gave me all the daily needs I needed and for their understanding.

-KHELIFA Souhail

Abstract

Living in the age of data has made the world eager to find the right information quickly and efficiently. But the task became harder and harder over the years because of the huge amount of existing data! That is why, scientists had thought of the Multi-Documents summarizing, a technique that can help people from finding the most sufficient data in no-time by using newly made inventions such as Machine learning and Neural Networking.

Automatic summarization is the process of shortening a text document with software, to create a summary with the major points of the original document. Why it is Important? It can quickly extract accurate content and help reader understand large volume of information.
Our job in this thesis was to build and fine tune a model based on transformers neural network called **Pegasus** and compare our work with the work of last year.

We have built and fine-tuned a model of Pegasus that was proposed by the Google AI team in 10 Jul 2020 that suggested a new way of fine-tuning. Using Clustering Algorithms, we have preprocessed our data sets and got pertinent results. The Algorithm consisted of choosing the most pertinent sentences and concatenated several documents into one document, then got sentences from that document and compared each sentence with all others.

The evaluation was done automatically using the ROUGE scores. Our method, as simple as it is, has shown promising results since the scores were higher then the previous works.

Key Words :
Text Summary, Transformers, BERT, GPT-3, Pegasus, ROUGE.

Résumé

Vivre à l'ère des données a rendu le monde désireux de trouver les bonnes informations rapidement et efficacement. Mais la tâche est devenue de plus en plus difficile au fil des années en raison de l'énorme quantité de données existantes ! C'est pourquoi, les scientifiques avaient pensé au résumé de Multi-Documents, une technique qui peut aider les gens à trouver les données les plus suffisantes en un rien de temps en utilisant de nouvelles inventions telles que l'apprentissage automatique et les réseaux neuronaux.

Le résumé automatique est le processus de raccourcissement d'un document texte avec un logiciel, pour créer un résumé avec les points principaux du document original. Pourquoi est-ce important? Il peut extraire rapidement un contenu précis et aider le lecteur à comprendre un grand volume d'informations. Notre travail dans cette thèse était de construire et d'affiner un modèle basé sur le réseau de neurones Transformers appelé **Pegasus** et de comparer nos travaux avec ceux de l'année dernière.

Nous avons construit et peaufiné un modèle de Pegasus qui a été proposé par l'équipe Google AI le 10 juillet 2020 qui suggérait une nouvelle façon de fine-tuning. En utilisant des algorithmes de clustering, nous avons prétraité nos ensembles de données et obtenu des résultats pertinents. L'algorithme consistait à choisir les phrases les plus pertinentes et à concaténer plusieurs documents en un seul document, puis à récupérer les phrases de ce document et à comparer chaque phrase avec toutes les autres.

L'évaluation a été faite automatiquement en utilisant les scores ROUGE. Notre méthode, aussi simple soit-elle, a montré des résultats prometteurs puisque les scores étaient supérieurs aux travaux précédents.

Mots clés : Résumé du texte, Transformers, BERT, GPT-3, Pegasus, ROUGE.

الملخص

لقد جعل الحيّز في عصر البيانات العالم حريصًا على العثور على المعلومات الصحيحة بسرعة وكفاءة. لكن المهمة أصبحت أصعب وأصعب على مر السنين بسبب الكم الهائل من البيانات الموجودة! لهذا السبب ، فكر العلماء في تلخيص المستندات المتعددة ، وهي تقنية يمكن أن تساعد الأشخاص في العثور على البيانات الأكثر كفاية في أي وقت من الأوقات باستخدام الاختراعات الحديثة مثل التعلم الآلي والشبكات العصبية.

التلخيص التلقائي هو عملية تقصير مستند نصي باستخدام برنامج لإنشاء ملخص بالنقاط الرئيسية في المستند الأصلي. لماذا من المهم؟ يمكنه استخراج محتوى دقيق بسرعة لمساعدة القارئ على فهم كمية كبيرة من المعلومات. كانت مهمتنا في هذه الأطروحة هي بناء وضبط نموذج يعتمد على شبكة محولات عصبية تسمى **Pegasus** ومقارنة عملنا بعمل العام الماضي.

لقد قمنا ببناء وضبط نموذج **Pegasus** الذي اقترحه فريق Google AI في 10 يوليو 2020 والذي اقترح طريقة جديدة للضبط.

باستخدام خوارزميات التجميع ، قمنا بمعالجة مجموعات البيانات الخاصة بنا مسبقًا وحصلنا على نتائج ذات صلة. تألفت الخوارزمية من اختيار الجمل الأكثر صلة وسلسلة العديد من المستندات في مستند واحد ، تم الحصول على جمل من هذا المستند ومقارنة كل جملة مع جميع الجمل الأخرى.

الكلمات الدالة:

ملخص النص ، Transformers، BERT، GPT-3، Pegasus، ROUGE

Contents Table

Figures Table

Grid Table

Introduction

Work context

Nowadays, finding the right data has become a struggle for the majority of people.

This problem itself came from the huge amount of data already existing. We can find thousands and thousands of same topics that are kind of repetitive, for example : a simple task like finding the right apple pie recipe has become a real drudgery for some, especially those who do not know how to use a search engine.

Problematic

Since we are in the era of speed, people are looking the right data in no time, sacrificing time in any terms is a huge lost.

That is why, summarizing has become an important tool.

But how can we summarize a lot of documents, how can we know if a document is pertinent or not ? !

With heavy and strong technologies, we built an application that solve these problems cited above ! During this thesis, we tested and built an app that can multi-summarize using Pegasus.

But the real question, how does it work exactly ? !

Work Objective

Our work during this project is to build a summarizing application and to understand how does it work exactly.

Machine Learning and Neural Networking are a crucial technologies which were required in our project. We dedicated the first chapter in understanding the Machine learning.

Our work will consist of :

— Understanding and training a model for summarizing.

— Building an application that can actually summarize.

— Compare and Evaluate our results to the previous ones.

Thesis Organisation

To make it simple and well organized we decided to split the work into four chapters which are :

— Chapter ONE : We will introduce the newly technologies used for summarizing.

— Chapter TWO : We will introduce the history and different types of Summarizing.

— Chapter THREE : Introducing the application that we've built for summarizing.

— Chapter FOUR : in this chapter we are going to do some experiments, meaning that we will analyse and compare our work to other's.

Chapter 1

Technologies used for Summarizing

1.1 Introduction

summarizing is an important art for humanity, useful to get the most important information and, with summarizing : a person can economize a big amount of time.
But regardless of its importance, it could take a lot of time to acquire a proper summarizing nowdays because of the massive data we're having in this era.

Machine learning and Neural Networking helps summarizing to make it quicker and more pertinent .

1.2 Machine Learning

1.2.1 Definition

Machine learning is a field of artificial intelligence based on : Logic and probability and Neural Networks. [1]

Machine learning is the study of algorithms and mathematical models that computer systems use to progressively improve their performance on a specific task. Machine Learning Algorithms build a mathematical model of sample data, known as "**training data**", in order to make predictions or decisions without being explicitly programmed to perform the tasks.

Training Data is a concept given to the massive dataset used to teach our machine learning models.

1.2.2 Types of Machine Learning

There are two noticeable types of Machine Learning which are called : **Supervised** and **unsupervised** Learning.

— First of, we have the **Unsupervised learning** : the unsupervised learning can be understood like giving a simple puzzle to a machine but without giving instructions, the machine has to build and guess the final image. It might get the image right but he has to put and repeat each task using different pieces of the puzzle.

— The **supervised learning** is approximately similar to the unsupervised in terms of the machine and the puzzle, the only difference is the machine will have a supervisor. Whenever it tries to put a piece of the puzzle, the supervisor will see if that's a correct piece and will allow it, if not then it redo the process. [2]

— **Supervised Learning** : In supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels. [3]

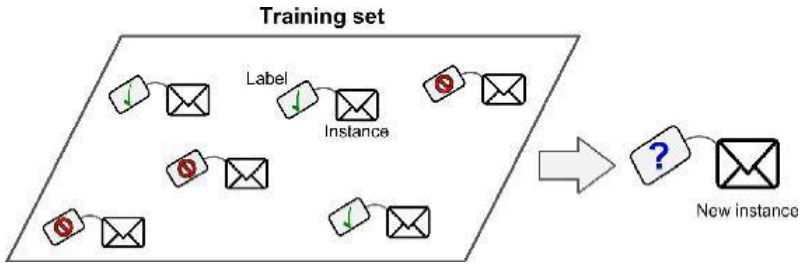

Figure 1.1 :A labeled training set for supervised learning [3]

One of the important tasks of supervised learning is **classification** and **Regression**

— **classification** : classification can be simply described as separating data into specific categories, like separating peaches from apples or egg from milk in real life.

— **Regression** : is a type of supervised learning method that uses an algorithm to understand the relationship between dependent and independent variables. This method is helpful to predict numerical values based on different data points.

— **Some Important Supervised Learning Algorithms** :

— K-Nearest Neighbour (K-NN).

— Linear Regression.

— Logistic Regression.

— Support Vector Machines (SVMs).

— **Unsupervised Learning** : In the unsupervised Learning. The Data is unlabeled thus, the system will have to learn without a teacher. Unsupervised Learning have three main tasks :

— **Clustering** : the unsupervised algorithm will try to put or to organize the unlabeled data in a same group depending on the similarity or differences of the data.

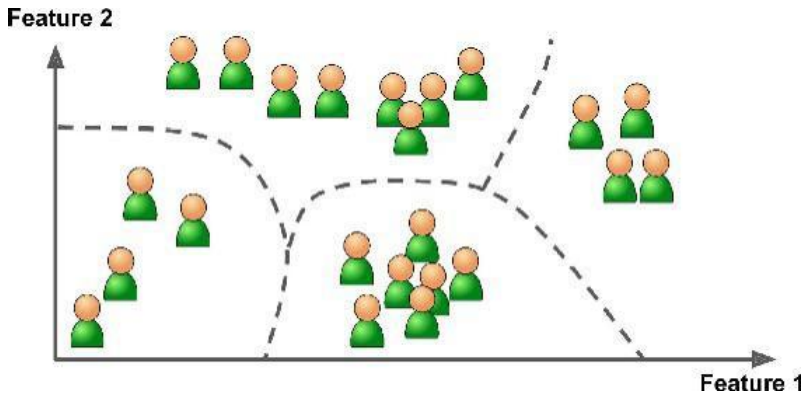

Figure 1.2 :Unlabeled datasets being clustered.[3]

— **Association** : the job of this method is to find an interesting attribute so it can be used in the later researchers. for example : those who buy shuffles and dirt have usually gardens in their home.

— **Dimensional reduction** : This method is used when the number of dataset is gigantic. It reduces the number of data inputs to a manageable size while also preserving the data integrity.

— **Some Important Unsupervised Learning Algorithms** :

— K-Means.

— Hierarchical Cluster Analysis (HCA).

— Expectation Maximization.

1.2.3 Clustering

As said earlier in the first chapter : Cluster analysis, or clustering, is an unsupervised machine learning task. It involves automatically discovering natural grouping in data. Unlike supervised learning (like predictive modeling), clustering algorithms only interpret the input data and find natural groups or clusters in feature space.[3]
There are many different algorithms of Clustering, in our cases : we have chosen to work with two algorithms.

1.2.3.1 Hierarchical Cluster Analysis (HCA)

Hierarchical clustering, also known as hierarchical cluster analysis, is an algorithm that groups similar objects into groups called clusters. The endpoint is a set of clusters, where

each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other.[18]

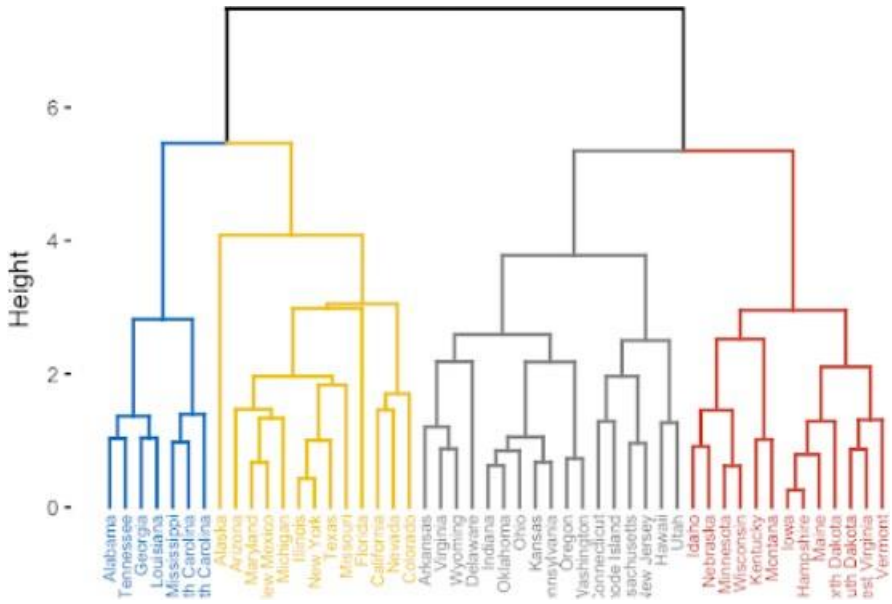

Figure 1.3 : Hierarchical Cluster Analysis[18]

1.2.3.2 Pairwise Clustering

Pairwise clustering methods partition a data set using pairwise similarity between data-points. The pairwise similarity matrix can be used to define a Markov random walk on the data points. This view forms a probabilistic interpretation of spectral clustering methods.[18]

1.2.3.3 K-Means

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into Kpre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different as possible.

1.2.3.4 How does it work

The way kmeans algorithm works is as follows :

— Specify number of clusters K.

— Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.[19]

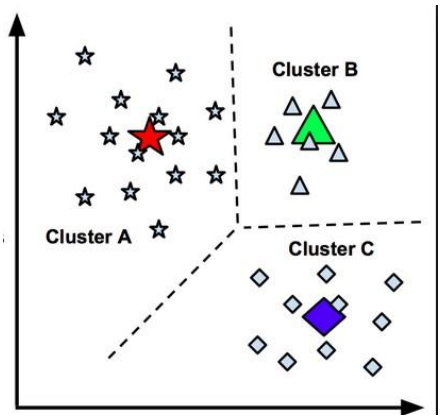

Figure 1.4 : Basic K-means Algorithm[18]

1.3 Neural Networking

A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes. Thus a neural network is an artificial neural network, for solving artificial intelligence (AI) problems.

1.3.1 Definition

a subset of **Machine Learning**, Neural Networks are the heart of Deep learning. A structure built to mimic the mechanism of an actual human brain. Neural Networks allow computer programs to recognize patterns and solve common problems in the fields of AI, machine learning, and deep learning. [5]

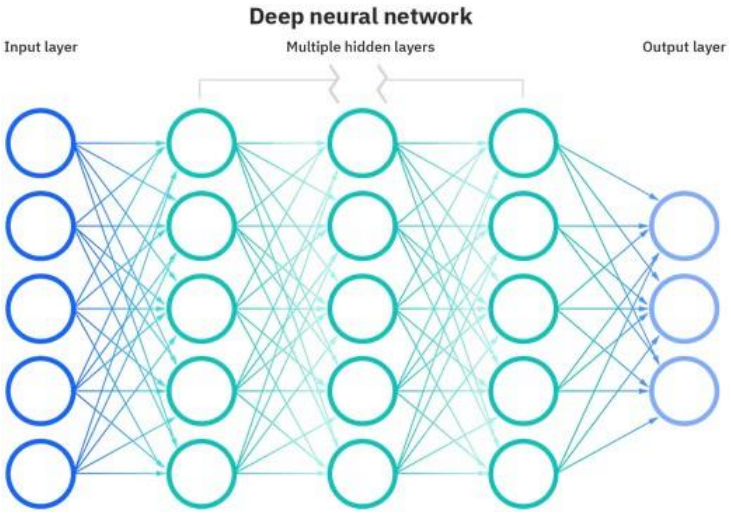

Figure 1.5 :A schema of Neural Networks[5]

1.3.2 How Do Neural Networks Work

As show in the figure above,a simple neural network includes an input layer, an output layer and, in between, a hidden layer. Those hidden layers are what makes the power of Neural Networks, inside them, the layers are connected with nodes and those connections are what form a "network". A node is patterned after a neuron in a human brain. Similar in behavior to neurons, nodes are activated when there is sufficient input. This activation spreads throughout the network, creating a response to the output. The connections between these artificial neurons act as simple synapses, enabling signals to be transmitted from one to another. Signals across layers as they travel from the first input to the last output layer and get processed along the way.

Once an input layer is determined, weights are assigned. These weights help determine the importance of any given variable. All inputs are then multiplied by their respective weights and then summed. The output is passed through an activation function, which determines the output. If that output exceeds a given threshold, it activates the node, passing data to the next layer in the network which will make the output our new input for another node. The process will continually keep on going depending on the data. This process is called : **feedforward network**.

$$\text{Input} = \sum_{i=1}^{m} w(i)x(i) + bias = w(1)x(1) + w(2)x(2) + w(3)x(3) + bias$$

$$output = f(x)\begin{cases} 1\ if\ \sum w(1)x(1) + b \geq 0 \\ 0\ if\ \sum w(1)x(1) + b < 0 \end{cases}$$

1.4 Different type of Neural Networks

There are many types and many methods of Neural Networks, we will briefly introduce the most known of them in our field which is Natural Language Processing.

1.4.1 Convolutional neural networks (CNNs)

1.4.1.1 Definition

Before developing this type of Neural Network, the supercomputer couldn’t detect simple images, it could beat a chess tournament but couldn’t have the ability to detect a simple puppy. That was a problem.

Convolutional neural networks (CNNs) emerged from the study of the brain’s visual cortex, and they have been used in image recognition since 1980s. In the last few years, thanks to the increase in computational power, the amount of available training data, training deep nets, CNNs have managed to achieve superhuman performance on some complex visual

tasks. They power image search services, self-driving cars, automatic video classification systems, and more. Moreover, CNNs are not restricted to visual perception : they are also successful at other tasks, such as voice recognition or natural language processing (NLP).

1.4.1.2 Functioning of CNNs

We will, in few points express how CNNs work :

— One Neural analyse only one visual range and capture Pixel brightness.

— Multiplies the brightness by a synaptic weights, (can be negative.)

— Keep only positive.

— Calculate brightness's sum.

— Synaptic weights will form a matrices : (Convolutional matrice. . .)

— These matrices are the ones who will be improved.

— Summarize the information of several neural into one info (Average Pooling) (Works better with Max Pooling.) Repeat "convolution-Pooling" again at the layer obtained after the "convolution pooling" operation.

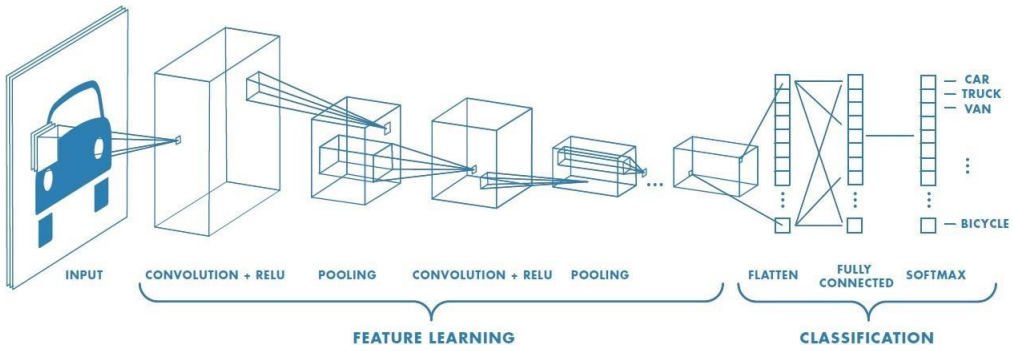

Figure 1.6 : Process of CNNs[5]

1.4.2 Recurrent neural networks (RNNs)

1.4.2.1 Definition

Loops allow us to solve many different problems, so it was more than logical to put a loop in a Neural Networking, and that is how Recurrent Neural Networks were born. The global idea of RNNs is : Implement synapses which will loop to recover information of past words for a better present understanding. for example : In French Language, the letter "Q" is usually followed by a letter "U".

1.4.2.2 Functioning of RNNs

use sequential information such as time-stamped data from a sensor device or a spoken sentence, composed of a sequence of terms. Unlike traditional neural networks, all inputs

to a recurrent neural network are not independent of each other, and the output for each element depends on the computations of its preceding elements. We will use an example to explain all of this : Let's take for example the input sentence "S4A"

— S will be read by the neural and will guess that it probably means Sun.

— 4 will be read and will be combined by the info "Sun" to understand the meaning of 4. . . etc. (This method resembles the Hidden Markov Model in Speech Recognition.).

— RNN is quite hard to train because of the "Loop instability" (Random multiplication of synapses which will lead to a "Vanishing Gradient." (vanishing or exploding. . .)

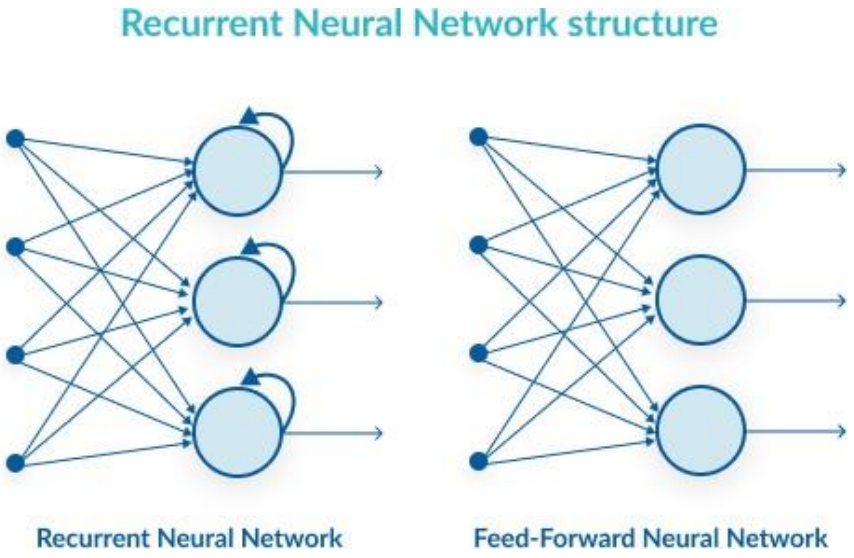

Figure 1.7 : Difference between FeedForward Network and RNN[6]

1.4.2.3 LSTMs

To avoid the "vanishing Gradient" problem a new approach has been introduced to the RNN which is the LSTM. It is a special kind of RNN. Which is capable of learning long term dependencies thus treating the problem of short term dependencies of a simple RNN. It is not possible for a RNN to remember the to understand the context behind the input while we try to achieve this using a LSTM.[6]

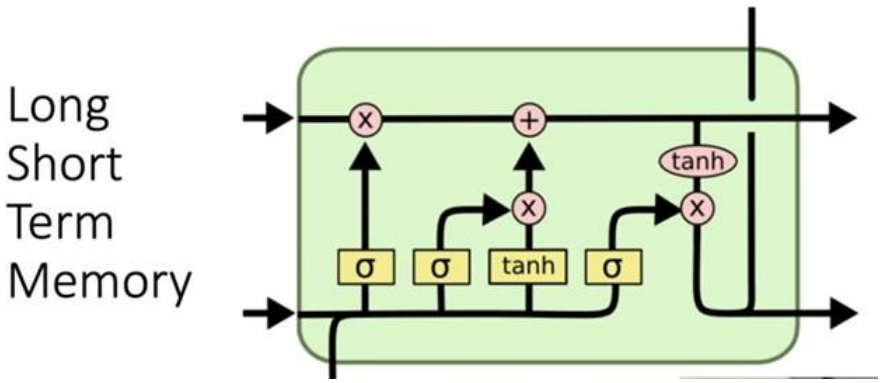

Figure 1.8 : LSTM's structure[6]

1.4.2.4 Gated recurrent unit (GRU)

GRU (Gated Recurrent Unit) aims to solve the vanishing gradient problem which comes with a standard recurrent neural network. GRU can also be considered as a variation on the LSTM because both are designed similarly and, in some cases, produce equally excellent results.

1.4.2.5 Functioning of GRU

To solve the vanishing gradient problem of a standard RNN, GRU uses, so-called, **update gate and reset gate**. Basically, these are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction.

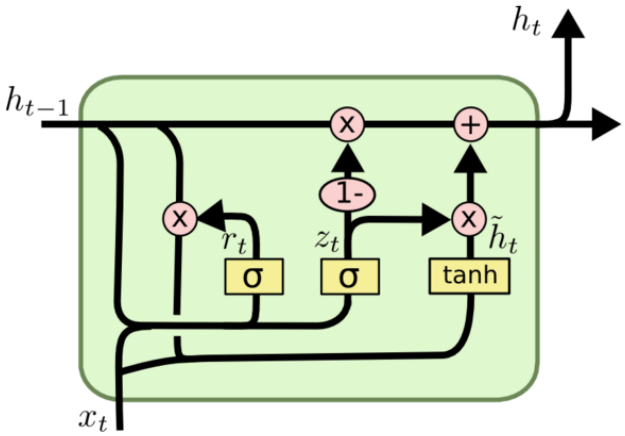

Figure 1.9 : Gated Recurrent Unit's structure[6]

1.4.3 Generative Adversary Network (GAN)

1.4.3.1 Definition

The concept of GANs was inspired by the Imitation game or Test of Turing that is still be used nowadays. It consists in creating fake data or imitating data that has been introduced, the data can be either : texts, images, sound or videos.

1.4.3.2 Functioning of GANs

This Neural Network looks like the relationship between a teacher and a student, what do we mean by that ?! We insert Two Neural Networks **Generative** and **Adversary**, here's the simplified steps of GAN's Networks :

— Inserting a Neural Network Adversary to a Neural Network Generator of fake data.

— The “Adversary” plays the role of a guide to the “Generator” and shows it what it is doing wrong and it is doing right. (The word “Adversary” does not literally mean adversary, on contrary, this adversary works as a teacher to a student.)

— It is the Adversary Neural Network that will be trained with Real data and will show the generator how to build Fake data.

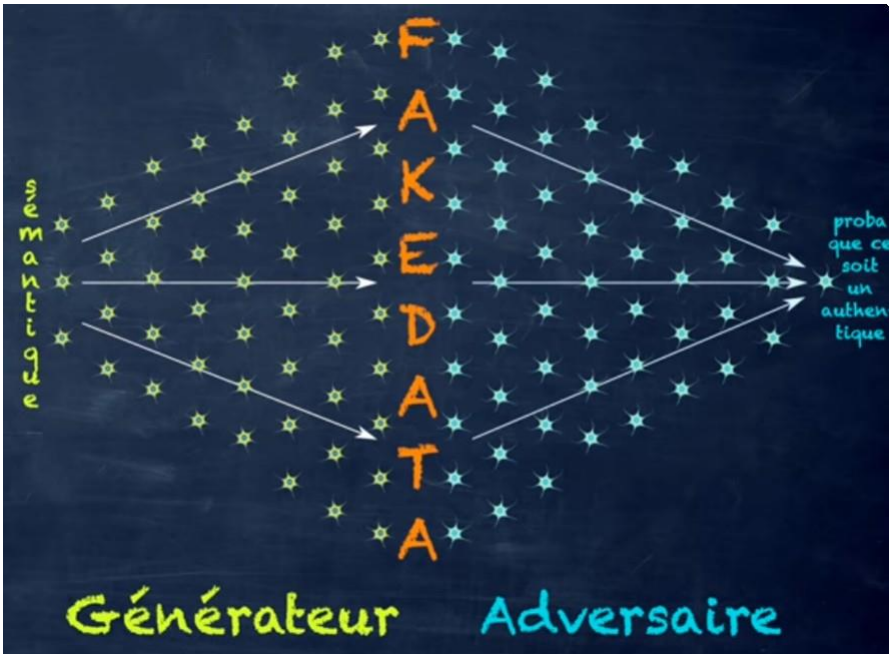

Figure 1.10 : Structure of GAN’s[40]

The Challenge of GANs : GANs are a complete Unsupervised learning, thus, it needs a huge amount of Data. It uses the Equation of Bayes, which is quite hard (almost impossible) to calculate, so it calculates the approximate results of this equation.

$$\mathbb{P}[T|D] = \frac{\mathbb{P}[D|T]\mathbb{P}[T]}{\mathbb{P}[D|T]\mathbb{P}[T] + \sum_{A \neq T} \mathbb{P}[D|A]\mathbb{P}[A]}$$

Figure 1.11 : Bayes Equation[40]

1.4.4 Autoencoder neural networks

1.4.4.1 Definition

Autoencoders are artificial neural networks capable of learning efficient representations of the input data, called codings, without any supervision. These codings typically have a much lower dimensionality than the input data, making autoencoders useful for dimensionality reduction. More importantly, autoencoders act as powerful feature detectors, and they can be used for unsupervised pretraining of deep neural networks. Lastly, they are capable of randomly generating new data that looks very similar to the training data ; this

is called a generative model. For example, you could train an autoencoder on pictures of faces, and it would then be able to generate new faces. Surprisingly, autoencoders work by simply learning to copy their inputs to their outputs. [9]

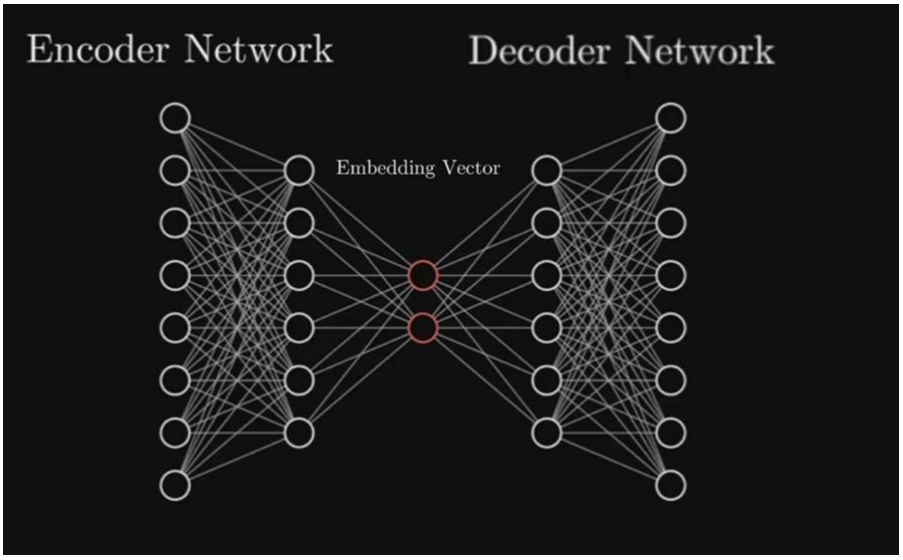

Figure 1.12 : Structure of an AutoEncoder Neural Network[39]

1.4.4.2 Functioning of Autoencoders

Auto Encoders are divided into two sides, Encoder and Decoder :

— **Encoder** : The Encoder's job is to transform the original input into a lower dimension representation. Ex : let's take (city, country), we might have : (Paris, France) (Algiers, Algeria) (Berlin, Germany) are acceptable. But : (Moscow, China) isn't, and that is because of a certain dimensional representation.

— **Decoder** : The Decoder's job is to recreate the original input using the output of the encoder, in other words : It tries to reverse the encoding process.

To put it simple : We force the encoder and the decoder to work together to find the most efficient way to condense the data into a lower dimension.

— **Important Note** : It is important to have an information loss between the encoder and decoder so we can avoid multiplying the input x1 and have a perfect output. Comes the "De-noising Autoencoders" that add noise to the data before passing the input into the network.

1.5 Transformers

1.5.1 Introduction

Attention is All You Need came out in late 2017. The paper announced a new Neural Network "model" that could change the game of neural networking.

1.5.2 Definition

Transformers are an encoder-decoder architecture. both encoder and decoder are divided into smaller parts that are called sub-layers The encoder has two sub-layer : a self attention layer that helps the encoder look at other parts as it encodes a specific word; and a feed-forward network. [8]

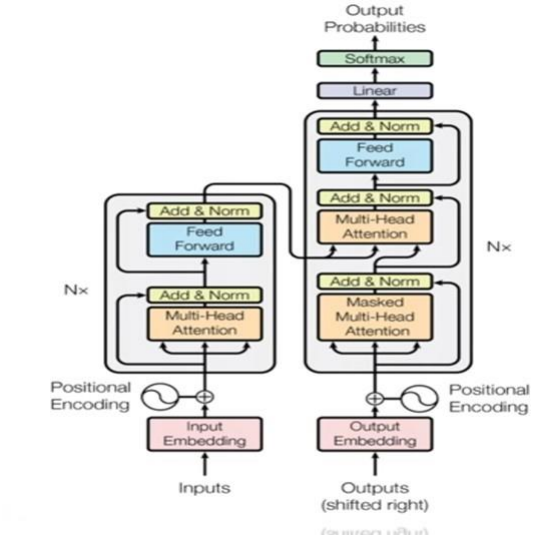

Figure 1.13 : Architecture of Transformers.[8]

What does the Encoder and Decoder of the Transformers consist of exactly ?

— **Encoder** : The encoder is composed of a stack of N = 6 identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is **LayerNorm(x + Sublayer(x))**, where Sublayer(x) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension d model = 512. [8]

— **Decoder** : The decoder is also composed of a stack of N = 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

— **Attention** :An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. [8]

1.5.3 Functioning of Transformers

We shall start with the **Encoder** part, we previously said that they have 6 layers, once again, we will explain them briefly :

1.5.3.1 Encoder

Each layer of the encoder stack has the following structure :

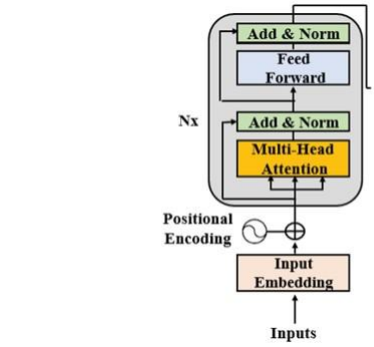

Figure 1.14 : A layer of the encoder stack of the Transformer[8]

The First Sub-Layer that we will talk about is the **Input Embedding**

— **Input Embedding** : The input embedding sub-layer converts the input tokens to vectors of dimension dmodel = 512 using learned embeddings in the original Transformer model.[8]

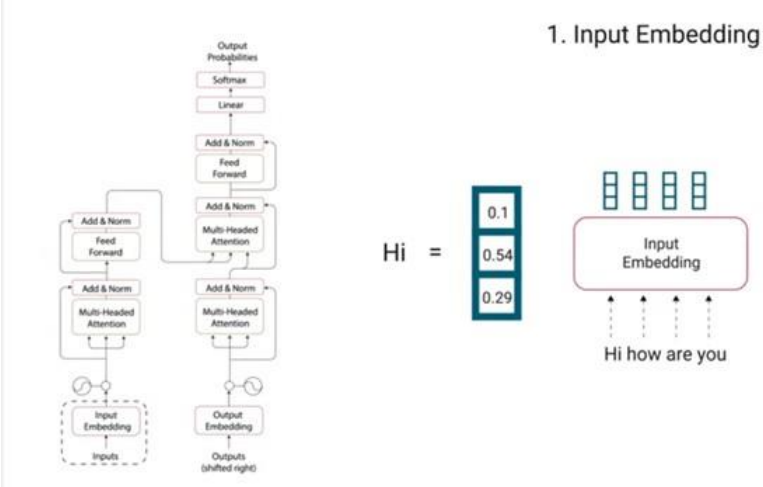

Figure 1.15 : Input Embedding Transforming a token into Vector[42]

— **Positional Encoding** : We enter this positional encoding function of the Transformer with no idea of the position of a word in a sequence. The idea is to add a positional encoding value to the input embedding instead of having additional vectors to describe the position of a token in a sequence. In other terms : Transformers don't have recurrence like RNN so we must add information about the position into the input embedding. . . [8]

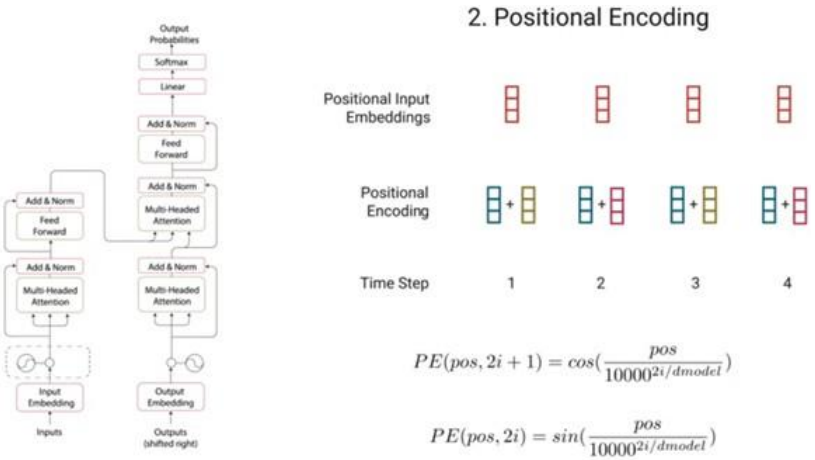

Figure 1.16 :Positional Encoding adding Position to Vectors[42]

— **Multi-Headed Attention** Multi-headed attention is a module in transformer neural network that computes the attention weights for the input and produces an output vector with encoded information on how each word should attend to all other words in a sequence. [14]

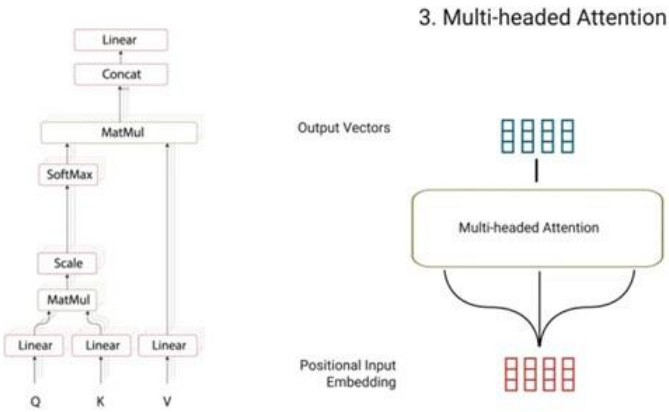

Figure 1.17 : Multi-Headed Attention[42]

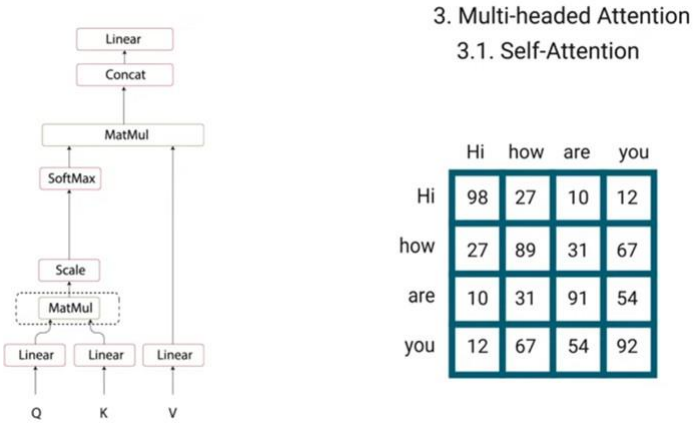

Figure 1.18 : Multi-Headed Attention Final OutPut[42]

The final sub-layer represents a softmax function that gives the probability of including a sentence in the summary.

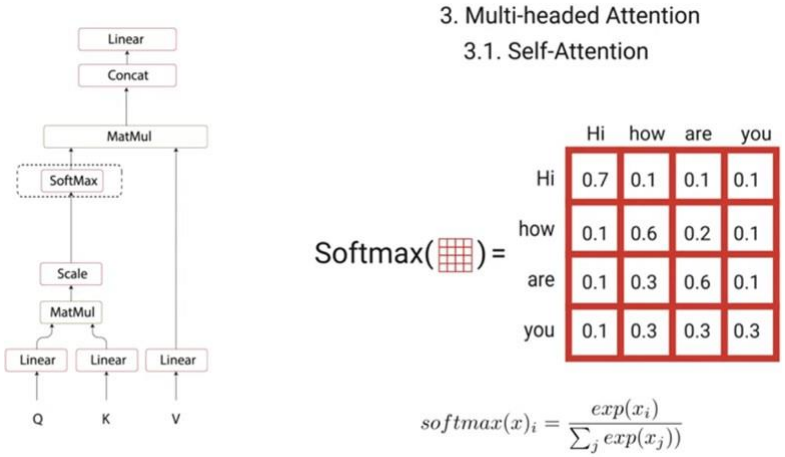

Figure 1.19 : Multi-Headed Attention SoftMax Function[42]

— **Residual connection, layer Normalization and Point Wise Feed Forward.** : One last step the encoder has are those three. they respectively mean :

— **Residual Connection** : The Multi-Headed attention output is added to the original input

— **Layer Normalization** : The Out put of the Residual connection goes through a layer Normalization.

— **Point Wise Feed Forward** : The out put of the layer normalization goes through a point wise feed forward network for further processing.

1.5.3.2 Decoder

The Decoder goes with the same steps as the encoder with slight changes such as “Mask Multi-Headed” which will mask or hide the future data of the word. Ex : in the sentence “I

am Fine.” The word “am” is only capable of accessing I, and cannot access Fine that what is called **Masked Multi-Head**

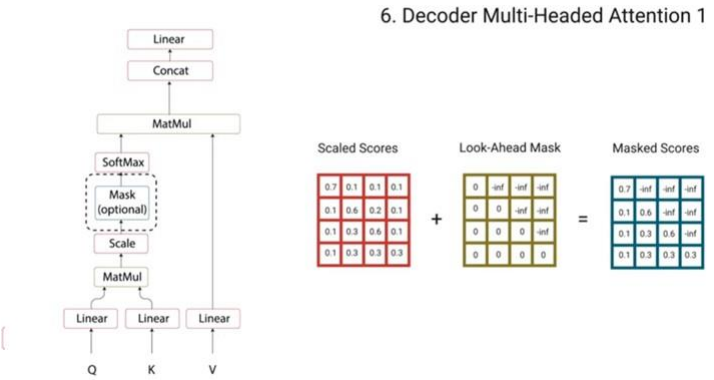

Figure 1.20 : Masked Multi-Headed Attention[42]

1.5.4 BERT

Bidirectional Encoder Representation from Transformers, Also known as BERT are a pre-trained language model built on top of the Transformer blocks. BERT can have either 12 or 24 layers, and each layer is a Transformer encoder. BERT is unique because it reads words from both directions at once. which will allow him to use different strategies such as : Masked Langage Model (MLM), and Next Sentence Prediction (NSP). BERT is composed by two phases : **Pre-Training** and **Fine Tuning**

— **Pre-Training** : Luckily for us : BERT is already pre-trained, all we need to do is to implement and fine tune our model into what we want it to do. In this instance : Summarizing.

— **Fine Tuning** : BERT can be used for multiple downstream NLP tasks. BERT has already a language understanding ability, it is easier to tune the weights. BERT can be used for classification, question answering, and any other task where the prediction at a certain position is allowed to look at other information from both directions.

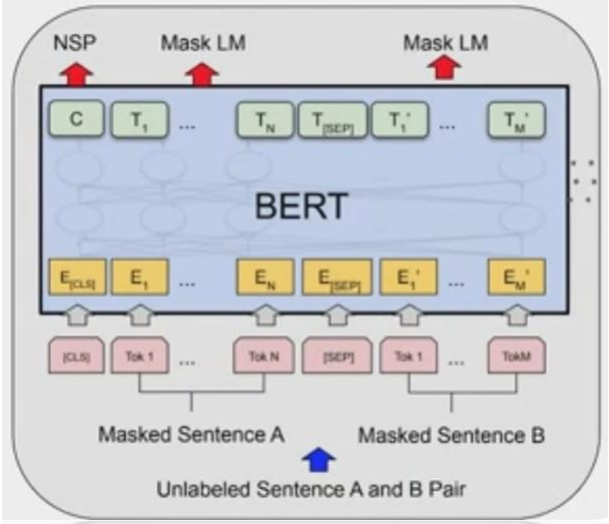

Figure 1.21 : BERT Structure[8]

1.5.5 Generative Pre-Trained Transformrer : GPT-2

GPT-2 is a model based on GPT and is a large transformer based language model that consists of solely stacked decoder blocks from the transformer architecture. Like any traditional language model, GPT2 outputs one token at a time. After each token is generated, that token is added to the sequence of inputs, and that sequence is fed to the model as an input, and so on. There are many variations of GPT-2, each one of them has more layers and more attention heads than the original Transformer decoder. [13]

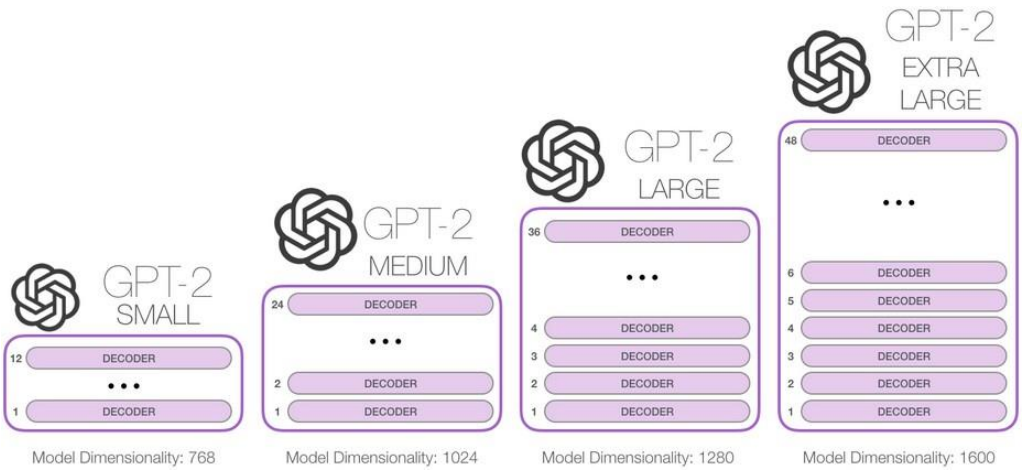

Figure 1.22 : Representation of GPT-2[42]

The Figure bellow demonstrates the main difference between BERT And GPT-2

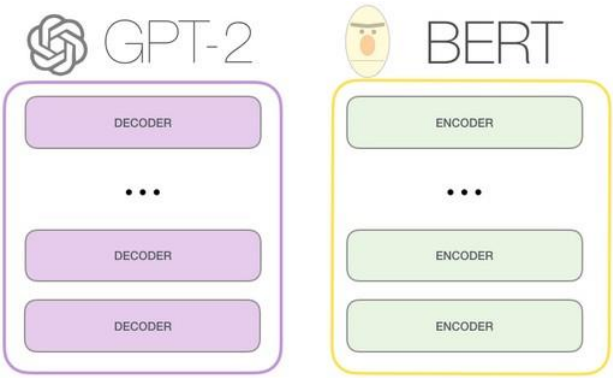

Figure 1.23 : Difference between GPT-2 and BERT[42]

1.5.6 Generative Pre-Trained Transformrer : GPT-3

GPT-3 or Generative Pre-trained Transformer is a is an autoregressive language model that uses deep learning to produce human-like text. GPT-3 was announced in the 28th of May in 2020 by OPEN-AI. GPT-3 is the largest trained Language model, with almost 175 Billion of parameters. [12]
GPT-3 is the third-generation language prediction model in the GPT-n series. It is part of a trend in natural language processing (NLP) systems of pre-trained language representations.
The quality of the text generated by GPT-3 is so high that it can be difficult to determine whether or not it was written by a human, which has both benefits and risks.
Sixty percent of the weighted pre-training dataset for GPT-3 comes from a filtered version of Common Crawl consisting of 410 billion byte-pair-encoded tokens. Other sources are 19 billion tokens from WebText2 representing 22 percent of the weighted total, 12 billion tokens from Books1 representing 8 percent, 55 billion tokens from Books2 representing 8 percent, and 3 billion tokens from Wikipedia representing 3 percent. GPT-3 was trained on hundreds of billions of words and is capable of coding in CSS, JSX, Python, among others.

Because of its capacity, thirty reaschers of the Open AI has deemed that GPT-3 is dangerous to be left as an Open Source. On September 22, 2020, Microsoft had annouced that it had licensed "exclusive" use of GPT-3. GPT-2 can still be used by the public to receive output, but only Microsoft has access to GPT-3's underlying model.

1.6 Conclusion

Machine Learning has made a huge step for science, it is in constant evolution. It is getting exponentially better over time.

Machine learning is nothing new, but the past years have been the biggest leaps and bounds in terms of advances in automatic text summarization using this approach. Factors like growing volume and massive data available, cheaper, more powerful com- putational processing, and affordable data storage made it only natural to quickly and automatically produce models that can process bigger, more complex data, and deliver faster, more accurate results, even on a very large scale. Many architectures can be used and combined for our task although recent models using Transformers seem to have greater potential.

In the Next Chapter we will be introducing the art of summarizing and its different types.

Chapter 2

A Brief History of Summarizing and its different types

2.1 Introduction

Summarizing, the art to transform a long text into a much shorter but pertinent paragraph.

Data explosion or the century of big data is what defines this era the most; what we're witnessing is only natural given the number of users and the variety of their concerns. As a consequence of this extraordinary growth, people are getting overwhelmed, first because of the expanding availability of information and secondly because of the little amount of time available to process it and make the best out of it ; that's where the summaries come.

Summaries are important when it comes to processing tremendous amounts of information. Among the different types of data, we are particularly interested in text data, in this case, a summary is defined as a text that is created from one or more writings, that conveys important information within the unique text, and that is no longer than half of the original text.

This chapter covers the general concepts of automatic text summarization. We will first give a general definition of automatic text summarization. Next, we will present the different types of automatic summaries, we will also explain the steps for generating a summary.

2.2 Definition

To put it simple. Summary is to retell or rewrite a passage leaving the main ideas. To be good at summarizing, all you need to do is to ask yourself :" **What are the most important details** needed for an understanding of a document. The challenge of Summary

overall is either you have **too much information** or **too little**. We need to be really careful of the information needed.
Summarizing is useful in many types of writing and at different points in the writing process. It is used to support an argument, provide context for a paper's thesis, write literature reviews, and annotate a bibliography. The benefit of summarizing lies in showing the "big picture," which allows the reader to contextualize what you are saying. In addition to the advantages of summarizing for the reader.
The explosion of data had made the summarizing task for human beings almost impossible ! And yet, its importance was high. We couldn't take away summarizing.
Thankfully, with the expansion of technology, scientists were capable of training and developing models that would eventually be able to make this task easier. That is what we call **Automatic Summarizing**.

2.3 Automatic Summarizing

Automatic Summarizing is the process of shortening a set of data using computers to create a subset with the most important information within the original content.
But it is obviously insufficient, we must dig deeper into the topic to completely understand the magic behind the curtains. And to do that, we will be explaining some terms.

2.4 Automatic Summarizing Process

To create a pertinent Automatic Summary, the automatic summarizing undergo three stages.

2.4.1 Pre-Processing

Pre-processing of a text is a set of steps to make a document in a format that is predictable and analyzable. In any input text, some words and symbols have no significant meaning related to the topic discussed and generally used to link words with each other. The repeated occurrence of these words can mess with the score of the important words. To solve this problem several methods have been used like Tokenization, Normalization and others.

2.4.1.1 Tokenization

Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens.

The goal of Tokenization is to split a large text into a set of sentences and then split those sentences to explore the words in it, therefore the list of tokens becomes an input for many statistical approach algorithms.

2.4.1.2 Normalization

Normalization is a set of steps to render the text in a standard form [17]. Normalization is highly recommended to have a standardized meaning of the texts. We can achieve the standard form of a text by applying certain operations such as :

— The removal of special characters and tags like HTML tags and other symbols.

— The Replacement of abbreviations and symbolized meanings with the correct words.

— Correction of the common misspelling mistakes.

2.4.2 Summary Generation

The process of generating summaries relies on many factors such as the format of the input, the purpose of the summarization, the type of summary. Because of that, there is no fixed method for generating a summary. Multiple solutions exist and follow a certain approach.

2.4.3 Post-Processing

Post-processing is a step that is generally applied to improve the quality and readability of the generated summary. It can remove the redundancy and resolve any inconsistencies concerning the pronouns. **Redundancy** elimination concerns either the removal of repetitive sentences in terms of meaning or the removal of consecutive repetitive words in the same sentence, resulting in a more compressed and clear summary. Post-processing techniques are not always used compared to the prepossessing ones which are usually crucial for this task.

2.4.4 Evaluation

Evaluating a summary is a difficult task because there is no ideal summary for a given document or set of documents. The goal of any summarization system is to optimize topic coverage and readability. there are many evaluation criteria:

— **Salience** : Are we capturing the most valuable information of the document ?

— **Length :** Is the summary of a proper length ?

— **Structure and coherence** : Summaries must be well structured and organized. A summary should not be just a block of information but should be assembled to form a coherent body of information.

— **Balance** : Is it covering all the topics from the initial document ?

— **Grammar** : Is the text grammatically correct ?

— **Non-redundancy**.

2.5 Types of Automatic Summarizing

2.5.1 Single-Documents Summarizing

In single document summarization, the system takes one document at a time and produces a summary for each input it takes. A single document can be composed of some sub-documents with multiple paragraphs. The described content of each of these sub-documents emphasis different aspects all surrounding the same topic.

2.5.2 Multi-Documents Summarizing

In multi-document summarization, we summarize multiple documents that have the same topic [25]. Using multi document summarization on documents obtained after a google search is another common use case. However, it is a big challenge to avoid redundancy since all of them are more likely to include a certain degree of similar information.

2.5.3 Abstractive Summrizing

Abstractive Text Summarization is the task of generating a short and concise summary that captures the salient ideas of the source text. The generated summaries potentially contain new phrases and sentences that may not appear in the source text.
Abstractive summarizers require a more profound analysis of the content, one major issue is that they are difficult to design due to their heavy dependence on linguistic techniques [17].

2.5.4 Extractive Summarizing

Extractive summarization consists of pulling out the most relevant parts of the documents, generally, sentences, that provides the idea of the subject to form the summary [28].
Extractive summarization is easier to perform because it just selects a subset of the input text based on its relevance. This ensures that the summary is readable and grammatically correct. However, consistency is not guaranteed. For example, if the summary system selects sentences containing references (acronyms, personal pronouns,etc.) and does not select sentences containing their antecedents, it’s very possible that the summary produced will be unclear. In addition, the extractive methods may produce summaries containing a

lot of redundancy.

Both Extractive and Abstracrive Summary have abundant methods used, we will briefly define some of the most important ones in the next highlight.

2.6 Automatic Summraztion Method

2.6.1 Statistical approach

The most well-known summarization approaches that use statistics are based on concept relevance and Bayesian classifier. This approach uses word frequency, uppercase words, sentence length, keywords, position in complete text, and phrase structure. this approach uses the TF-IDF (terms frequency approach) equation to summarize its documents. [29]

$$TF(t,d) = \frac{fd(t)}{\sum_{i=1}^{|d|} |ti|d}$$

— tf(t,d) is the term frequency of the term t in document d.

$$IDF(t) = log\ \frac{D}{\{dj{:}t\ i \in dj\}}$$

— |D| is the number of documents in the corpus.

— | d : t d| is the number of documents containing the word t.

The final Equation should look like this :

$$TF - IDF = TF * IDF$$

2.6.2 Graph approach

The nodes in graph based summarization approaches represent the sentences, and the edges represent the similarity among the sentences. The similarity values are calculated using the overlapping words or phrases. The sentences with highest similarity to the other sentences are chosen as a part of the resulting summary. TextRank and Cluster LexRank are two methods that use graph based approach for document summarization.

2.6.3 Abstractive Approach

Non-Extractive Summarization Methods (Abstractive summarization methods) try to fully understand the given documents, even non-explicitly mentioned topics, and generate new sentences for the summary. This approach is very similar to the way of human summarization. There are approaches that create summaries in a non-extractive manner, using information extraction, ontological information, information fusion and compression. It determines the quality of summary with respect length, coherence, structure, content.Two parameters are important in text summarization, the Compression Ratio, i.e. how much shorter the summary is than the original, and the Retention Ratio, i.e. how much of the central information is retained.A broad division into evaluation techniques would be Intrinsic and Extrinsic evaluation

2.6.4 Machine Learning approach

Machine learning can solve the problem of the previous approaches by combining features in the **statistical approach** for extracting the key sentences. We will dive deeper in the related works section.

2.7 Related Works

In this section, we will be talking about some of the previous works in that field, to the very beginning until last year's.

2.7.1 Mono-Document

2.7.1.1 Extractive Summaraziton

— **Statistical Approach** : The first work of automatic summarization follows the statistical approach. Baxendale [19] used a corpus of 200 scientific paragraphs to test the impact of the position of sentences in a paragraph on its importance; and he found that in 85 percent of cases, the first sentence is the most important for the paragraph. The last sentence is the most important in 7 percent of cases. He explained the result by supposing that topic sentences are more likely occur very early or very late in a document. He took the first and last sentence in each paragraph to form the summary. Despite its simplicity it was a kind of reasonable method for the type of documents he was trying to summarize. Luhn [40] proposed a more complex way to generate summaries and used a feature called term frequency which supposes that a term is important if it occurs many times. During the preprocessing, he removed stop words and stems so that it can objectively calculate the frequency of each word. He then grouped the sentences based on the concentration

of the keywords. The group which has the most significant words is used to score the sentence, and the summary is generated by extracting the highest score sentences. The main advantage of term frequency is that it is language-independent. However; it may consider some terms that do not thoroughly characterize the topic being addressed as being relevant. For instance, the words informatics and computer are frequent in computer science topics. Another problem is the case when some words are not so frequent but are supposed to be relevant. To solve the problem discussed above, Salton and Yang [19] proposed another feature called tf idf where tf refers to term frequency and idf represents the inverse document frequency. It expresses that a word is more important when it is more frequent in the analyzed document and not frequent in the corpus of analyzed documents.

Edmundson work used some of the already mentioned features ; he used the position of a sentence in a document in the same way as Baxendale [19], he also looked at term frequency, the same way as Luhn [19] ; but he also added a feature called cue words. Cue words such as our "results indicate", "in this article", etc. were used to determine relevant information to the summary. These words were manually selected and saved in a dictionary that comprises three sub-dictionaries : bonus words, stigma words, and null words. Bonus words include superlatives, adverbs, etc. and add value to the sentence. Stigma words are words that negatively affect sentence importance. They include anaphoric expressions, belittling expressions, etc. Finally, null words represent neutral or irrelevant words to the sentence and are often stop words.

Another thing he looked at was the structure of the document : is it a headline? Is it a title ? Is it the first sentence after a title ? As for the score of the sentence, he created a linear combination of these four features. Besides, Fattah and Ren defined positive cue words as words that have frequent occurrences, while negative words are the ones thatare most unlikely to be included in a summary. To achieve that, we calculate the score of a sentence given a word W and a summary S.

— **Graph approach** : Salton et al. constructed a graph of similarity using document paragraphs. Each paragraph represents a node that is connected to another when their similarity exceeds a given threshold. The authors defined a feature called bushiness which is the number of a node's connections. The most scored paragraphs in terms of bushiness are extracted to form a summary. The equation bellow describes the score based on the number of arcs, where G = S, A is the graph of similarities between the sentences, S is the set of sentences and A is the set of arcs.

$$\text{Score(arc)} = sj : a(si, sj) \ \ A/sj \ \ S, si =/= sj$$

This equation means that the score of any sentence si is the number of the nodes that sentence is linked to. Also, a(si, sj) A means that the sentence si has an arc with a sentence si, and si 6= sj makes sure that we do not count the arc linked to the same sentence. Thakkar, Dharaskar, and Chandak [20] proposed a method that uses the shortest path algorithm for the summary generation. First, they built a graph model for representing the text as a way to connect text entities in the graph to form meaningful relations. Then, they applied a ranking graph-based algorithm to score each vertex of the generated graph in the previous step. Finally, they used the shortest path algorithm on the graph to generate the text summary.

— **Compression approach** : Sentence compression aims to retain the most salient information of a sentence and delete the least critical information, rewritten in a short form. In Jing and Mckeown, the authors claimed that it is often used by professional summarizers. They found out that 78 percent of the summary sentences are taken from the input document, and more than half were compressed. Sentence compression works can be classified into two approaches : a sentence deletion approach and an abstractive approach. The deletion based approach is based on the removal of unnecessary parts of the input which is generally on a sentence level. That is, the result of the compression is a subsequence of the source sentence. Consequently, those systems produce an extractive summary. For instance, Jing work is one of the earliest ones in this direction. Hepresented a sentence compression system that uses several knowledge sources. The purpose is to compress by removing as many unnecessary parts of the sentences as possible without detracting from the main idea that the sentences convey. These sources include syntactic knowledge, context information, and statistics extracted from a corpus of professional summaries.

— **Machine learning approach** : As said earlier, the machine learing has solved many problems of the previous approaches.

Yatsko, Starikov, and Butakov tuned the features according to the input document's genre. They used forty-five parameters of various types : statistical, positioning, and discourse to identify those that are most meaningful for a given genre of the document. Then, using K-Means, grouped the documents in a corpus according to genres : scientific, press, and artistic. Finally, they used the corpus to identify the specific parameters for each genre by as signing a weight to each one. Hence, the system can distinguish the genre of the input document and execute the adequate model of scoring.

Wong, Wu, and Li proposed a learning-based approach to combine four criteria : surface features, content features, event features, and relevance features. Surface featuresbase on the structure of documents or sentences, content features measure a sentence based on content carrying words, event features represent sentences by events they contain, and relevance features evaluate a sentence from its relatedness with other sentences. After

examining feature vectors of sentences, they employed a supervised learning classifier. The system re-rank candidate sentences considering the length of the final summary that is fixed. Finally, they extracted the top sentences to generate the final summaries.

authors proposed a system that improves performance for both extractive and abstractive single-document summarization following the encoder-encoder-decoder paradigm. The extractive model classifies each sentence in a document as being summary worthy or not. To enhance the sequence classification process, they encoded the input document with a Transformer. The input is the concatenation of the vector representation of the document sentences. Each sentence representation is the average of the vector representation of its constituent words. The transformer encoder is a six stacked identical layers. The extraction model has a final softmax layer that gives the probability of including a sentence in the summary.

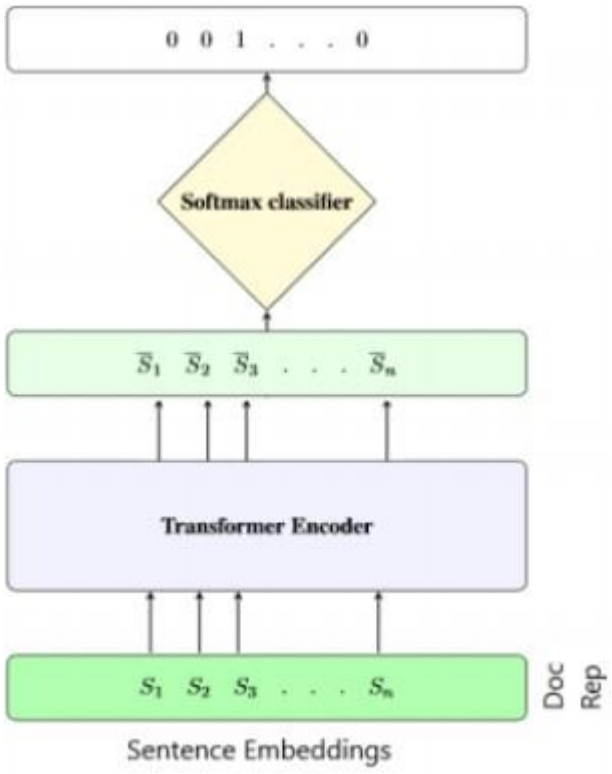

Figure 2.1 : The extractive model architecture[28]

Besides, the model called BERTSUM [31] is a variant of BERT (Bidirectional Encoder Representations from Transformers)for extractive summarization. The global architecture of this model comprises two components : a BERT large component, and a summarization component. BERT large consists of twenty-four transformer encoder layers. It expects an input of size 1024 and has sixteen head in the multi-head attention sublayer.

In vanilla BERT, The [CLS] is used as a symbol to aggregate features from one sentence or a pair of sentences. While in BERTSUM, they modify the model inserting external [CLS] tokens at the start of each sentence, and each [CLS] symbol collects features for the sentence preceding it. The vector T i which is the vector of the i th [CLS] symbol from the top BERT layer is used as the representation for sentence i.
Also since BERT has only two labels for indicating sentences, those labels are hard to use for extractive summarization to distinguish between sentences. Hence, they modify the segment embeddings and assign EA or EB depending on whether the sentence is odd or even. For example, a document with five sentences [sent1, sent2, sent3, sent4,sent5], would assign embedding [EA, EB, EA, EB, EA]. The intuition behind this modification is to learn document representations hierarchically where lower Transformer layers represent adjacent sentences, while higher layers, in combination with self-attention, represent multi-sentence discourse.

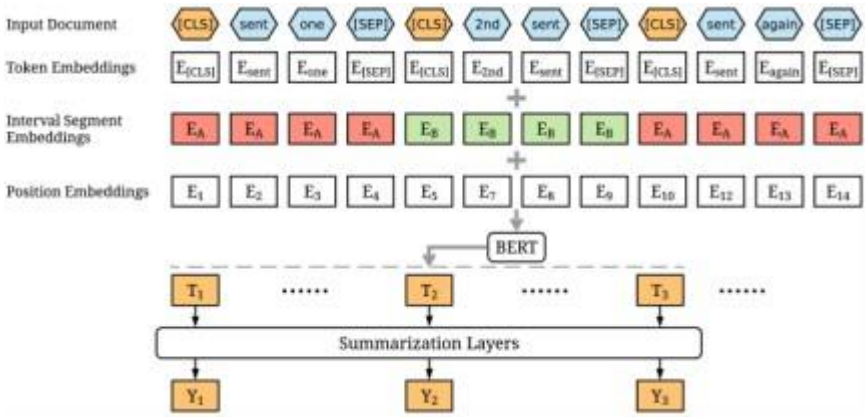

Figure 2.2 : BERTSUM Architecture[31]

They used the sentence representation from BERT as an input to other layers which are specific to summarization. Those layers will capture document-level features to generate the summaries. For each sentence, they calculate the final predicted score and compute the error using the Binary Cross Entropy of the generated label output Y i against the gold label Y i . These summarization layers are jointly fine-tuned with BERT.

2.7.1.2 Abstractive Summaraziton

— **Compression approach** : For the compression approach in abstractive summarization there are fewer works compared to extractive summarization. We can cite the work of Choi et al. [53] who proposed an abstractive sentence compression method using event attention for compression sentences of news articles. Event attention focuses on the event words of the source sentence in generating a compressed one. The attention score represents the

combination of the event attention and the global attention used to understand the global information of a sentence.

— **Machine Learning Approach** : Abstractive summarization systems make use of deep learning in multiple ways. Rush, Chopra and Weston implemented successfully deep learning to automatic text summarization by applying a local attention-based model to their summarizer called NAMAS. The attention-based model generates the summaries by taking into consideration all words when processing a particular word. Some of the limitations of their method are : it processes only documents with a size of around 500 words and generates very short summaries.

In the same direction, Nallapati et al. used an attention model as an encoder-decoder recurrent neural network (RNN). The encoder is a bidirectional GRU-RNN and the decoder is a unidirectional GRU RNN. The decoder has the same hidden size as the encoder and a softmax layer over the target vocabulary to generate words. To identify key concepts and entities in the document, they captured linguistic features such as part-of-speech tags, named-entity tags, and tf idf statistics of the words. Finally, for each word in the source document, they looked-up its embeddings from all of its associated tags and concatenated them into a single long vector, as shown in figure 2.5. On the target side, they used only word-based embeddings as the representation. As we can see, there is one embedding vector each for POS, NER tags, TF and IDF values, which are concatenated together with word-based embeddings as input to the encoder.

— **BART** :

BART is a denoising autoencoder that maps a corrupted document to the original document it was derived from. It is implemented as a sequence to-sequence model with a bidirectional encoder over corrupted text and a left-to-right autoregressive decoder. For pre-training, we optimize the negative log likelihood of the original document.

BART uses the standard sequence-to-sequence Transformer architecture from, except, following GPT, that we modify ReLU activation functions to GeLUs and initialise parameters from N (0,0.02). For our base model, we use 6 layers in the encoder and decoder, and for our large model we use 12 layers in each. The architecture is closely related to that used in BERT, with the following differences : each layer of the decoder additionally performs cross-attention over the final hidden layer of the encoder (as in the trans- former sequence-to-sequence model); and BERT uses an additional feed-forward network before word prediction, which BART does not. In total, BART contains roughly 10 percent more parameters than the equivalently sized BERT model.

— **Pre-training BART** :

BART is trained by corrupting documents and then optimizing a reconstruction loss—the

cross-entropy between the decoder's output and the original document. Unlike existing denoising autoencoders, which are tailored to specific noising schemes, BART allows us to apply any type of document corruption. In the extreme case, where all information about the source is lost, BART is equivalent to a language model. We experiment with several previously proposed and novel transformations, but we believe there is a significant potential for development of other new alternatives. The transformations we used are summarized.

— **Token Masking** : Following BERT random tokens are sampled and replaced with [MASK] elements.

— **Token Deletion** : Random tokens are deleted from the input. In contrast to token masking, the model must decide which positions are missing inputs.

— **Text Infilling** : A number of text spans are sampled, with span lengths drawn from a Poisson distribution (= 3). Each span is replaced with a single [MASK] token. 0-length spans correspond to the insertion of [MASK] tokens. Text infilling is inspired by Span-BERT, but SpanBERT samples span lengths from a different (clamped geometric) distribution, and replaces each span with a sequence of [MASK] tokens of exactly the same length. Text infilling teaches the model to predict how many tokens are missing from a span.

— **Sentence Permutation** : A document is divided into sentences based on full stops, and these sentences are shuffled in a random order.

— **Document Rotation** : A token is chosen uniformly at random, and the document is rotated so that it begins with that token. This task trains the model to identify the start of the document.

— **Fine-tuning BART** :

Sequence Classification Tasks : For sequence classification tasks, the same input is fed into the encoder and decoder, and the final hidden state of the final decoder token is fed into new multi-class linear classifier. This approach is related to the CLS token in BERT; however we add the additional token to the end so that representation for the token in the decoder can attend to decoder states from the complete input.

Token Classification Tasks : For token classification tasks, such as answer endpoint classification for SQuAD, we feed the complete document into the encoder and decoder, and use the top hidden state of the decoder as a representation for each word. This representation is used to classify the token.

Sequence Generation Tasks : Because BART has an autoregressive decoder, it can be directly fine tuned for sequence generation tasks such as abstractive question answering

and summarization. In both of these tasks, information is copied from theinput but manipulated, which is closely related to the denoising pre-training objective. Here, the encoder input is the input sequence, and the decoder generates outputs auto-regressively.

Machine Translation : We also explore using BART to improve machine translation decoders for translating into English. Previous work has shown that models can be improved by incorporating pre-trained encoders, but gains from using pre-trained language models in decoders have been limited. We show that it is possible to use the entire BART model (both encoder and decoder) as a single pretrained decoder for machine translation, by adding a new set of encoder parameters that are learned from bitext. More precisely, we replace BART's encoder embedding layer with a new randomly initialized encoder. The model is trained end-to-end, which trains the new encoder to map foreign words ino an input that BART can de-noise to English. The new encoder can use a separate vocabulary from the original BART model. We train the source encoder in two steps, in both cases backpropagating the cross-entropy loss from the output of the BART model. In the first step, we freeze most of BART parameters and only update the randomly initialized source encoder, the BART positional embeddings, and the self-attention input projection matrix of BART's encoder first layer. In the second step, we train all model parameters for a small number ofiterations.

— **Pegasus** : Pegasus, is a newly transformer structure, quite the same as its predecessors BERT and BART but stronger and more efficient. in the conception chapter we will be digging deeper into Pegasus and how does it work.

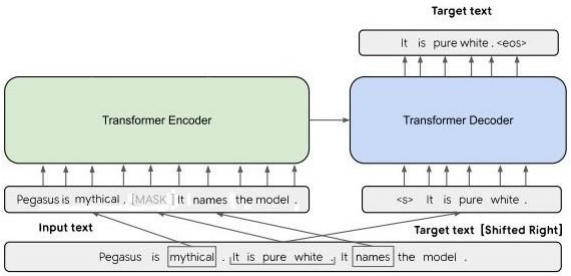

Figure 2.3 : Structure of Pegasus[28]

2.7.2 Multi-Documents Summarzation

There aren't many related works in that field. Last Year's work was the first to adapt a Multi-Documents Summaraztion[28]. They have used the GPT-2 structures to make theirworks and it gave interesting results.

They calculated the semantic similarity between a pair of sentences by taking the average of the word embeddings and calculated the cosine between the resulting embeddings.

Cosine similarity is a metric that performs an inner product between two vectors ; if two vectors are similar then the angle between them would be equal to 0 degree, therefore the cosine value would be equal to 1. And finally they extracted the features embeddings from the [CLS] token. To compute the similarity they : – Fed the first sentence into BERT : – Extracted the sentence embedding from the [CLS] token. – Performed the same two previous steps for the second sentence. – computed the cosine value between those two embeddings.

Since most recent works deal with abstractive summarization, we can conclude that researchers are finally starting to get familiar with this type of summarization.

2.8 Conclusion

In this chapter we have seen a brief history of the art of summarizing. We explained briefly some its techniques and its struggles over the years.

The Evolution of technologies has prevented natural summarizing to be efficient. The explosion of Data has made it almost impossible to natural summarize.

Luckily, the evolution of Machine Learning has found us a solution which is Automatic Summerzation. We have seen some of the previous works of researchers who excelled on that.

Now that all has been settled, we will be approaching the conception of our Application and the steps that were needed for us to build a good Summarizing application.

Chapter 3

Conception

3.1 Introduction

In the previous chapters, we have briefly introduced the definitions needed for our conception.
In this Chapter, we will be showing our final work : **Borgir, the application that summz almost anything**.
But first, we will be showing off the steps that were needed for a good process.

3.1.1 Problematic

As seen previously, all the related works of this topic (mutli-document summarizing) were complementary to each other, they reinforced their predecessors by removing the loopholes that they encoutered.

By using **Pegasus**, a transformer based architecture that was designed in 2020, we had better results comparing to the previous works. Our work showed promising results.
But before going into the experiments, we will be digging deeper in our work to have a better understanding of how it did work.

3.2 System Architecture : Pegasus

A pre-training large-based decoder models on massive text corpora with new self-supervised objectives, that's what it was needed to manage massive Data. And that is how Pegasus was made : A Massive Model that has 568Million parameters.
We will try to peel the steps of Pegasus to make it simple to understand.

3.2.1 Pre-Training

— A new pre-training objective was proposed in the Pre-Training of Pegasus, **GSG**, quite different than the Masked Language Model (MLM) of BERT. the Gap Sentences Generation (GSG) is a life changer.

— **Gap Sentences Generation** : Instead of masking few words, The Gap Sentences Generation mask a whole sentence. By selecting and masking whole sentences from documents, and concatenating the gap-sentences into a pseudo-summary. The corresponding position of each selected gap sentence is replaced by a mask token [MASK1] to inform the model. To even more closely approximate a summary, sentences that appear to be important/principal to the document are masked. The resulting objective has both the empirically demonstrated benefits of masking, and anticipates the form of the downstream task. There are 3 primary strategies for selecting m gap sentences without replacement from a document, D = xin,comprised of n sentences : **Random** Uniformly select m sentences at random. **Lead** Select the first m sentences. **Principal** Select top-m scored sentences according to importance.

— **Masked Language Model** : MLM is applied to train the Transformer encoder as the sole pre-training objective or along with GSG. When MLM is the sole pre-training objective, the Transformer decoder shares all parameters with encoder when fine-tuning on downstream tasks following. However, MLM did not improve downstream tasks at large number of pre training steps, so it was chosen to not to include MLM in the final model PEGASUSLARGE. The Figure bellow is a representation of the Gap Sentence Generation process.

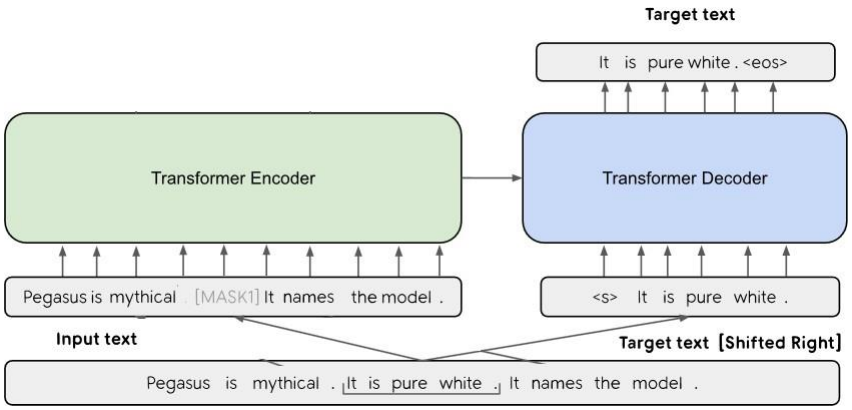

Figure 3.1 : Gap Sentence Generation [28]

N.B : The Masked Language Model has been taken away from the final model.

3.2.1.1 Pre-training Corpus

For the pre-training, two large text corpora were chosen :

— **C4** : The Colossal and Cleaned version of Common Crawl, introduced in Raffel et al. (2019) ; consists of text from 350M Web-pages (750GB).

— **HugeNews** : a dataset of 1.5B articles (3.8TB) collected from news and news-like websites from 2013-2019. A whitelist of domains ranging from high-quality news publishers to lower-quality sites such as high-school newspapers, and blogs was curated and used to seed a web-crawler. Heuristics were used to identify news-like articles, and only the main article text was extracted as plain text.
Pegasus was already pre-trained when we implemented it.

3.2.2 Downstream Tasks/Datasets

For downstream summarization, public abstractive summarization was used as datasets, a them through TensorFlow Summarization Datasets 1, which provides publicly reproducible code for dataset processing and train/validation/test splits. Here are the names of some famous datasets : CNN-Daily Mail. GigaWord. AESLEC. XSum. Multi-News.[10]

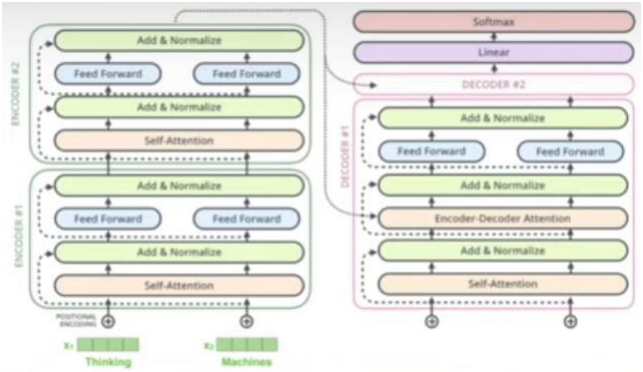

Figure 3.2 : detailed Pegasus Architecture[28]

3.2.3 Embedding

BERT and RoBERT has set a new state-of-the-art performance on sentence-pair regression tasks like semantic textual similarity. However, it requires that both sentences are fed into the network, which causes a massive computational overhead : Finding the most similar pair in a collection of 10,000 sentences requires about 50 million inference computations (65 hours). The construction of BERT makes it unsuitable for **semantic similarity** search as well as for **unsupervised tasks like clustering**. Because of that problem, Sentence-BERT (SBERT) was created, a modification of the pretrained BERT network that use siamese and triplet network structures to derive **semantically meaningful sentence embeddings** that can be compared using c**osine-similarity**. This reduces the effort for finding the most

similar pair from 65 hours with BERT / RoBERTa to about 5 seconds with SBERT, while maintaining the accuracy from BERT. [27]

3.2.4 Treatment

3.2.4.1 Fine-Tuning

Fine-tuning is a way of applying or utilizing transfer learning. Specifically, fine-tuning is a process that takes a model that has already been trained for one given task and then tunes or tweaks the model to make it perform a second similar task. **To makeit simple : Fine-Tuning is taking a pre-trained model and tweaking it to perform a second similar task, in this case : Summarizing. We fine-tuned our pre-trained model (pegasus) with several Datasets, which are : AESLEC, CNN-DailyMail, GigaWord, Multi-News.**

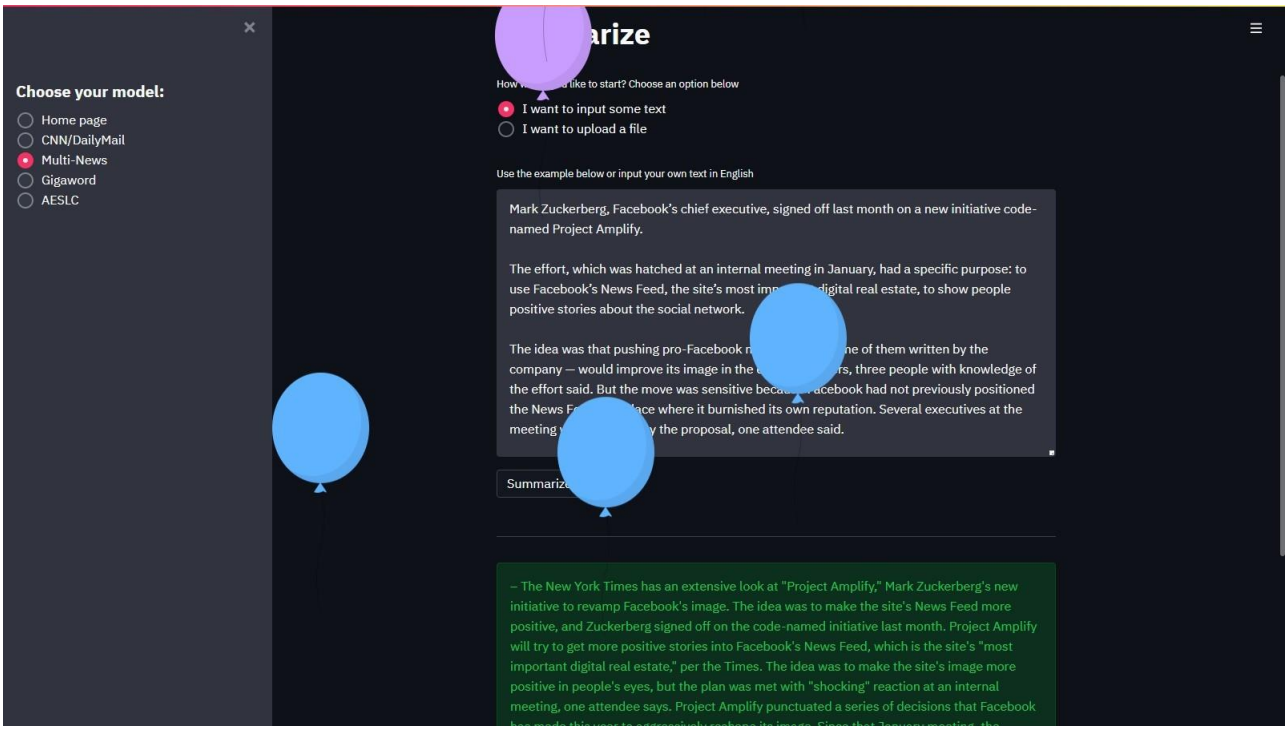

Figure 3.3 : An Overview of the Application

In the next section, we will be diving deeper into how we have built this application.

3.2.5 Borgir

3.2.5.1 Dependencies

To create a great application, we had to import some important libraries that are shown in the figure bellow :

— import streamlit as st.

— from transformers import PegasusForConditionalGeneration, PegasusTokenizer.

— from io import StringIO.

— import extractiveSummarization.

— **streamlit** : Streamlit is an open source app framework in python language. It helps us create beautiful web-apps for data science and machine learning in a little time. It is compatible with major python libraries such as scikit-learn, keras, pytorch, latex, numpy, pandas, matplotlib.

— **StringIO** : The StringIO module is an in-memory file-like object. This object can be used as input or output to the most function that would expect a standard file object. When the StringIO object is created it is initialized by passing a string to the constructor. If no string is passed the StringIO will start empty. In both cases, the initial cursor on the file starts at zero.[R]
To put it simple, StringIO helps us read files.

```
if source == 'I want to upload a file':
    files = st.file_uploader('Upload your file here', accept_multiple_files=True)
    input_su = ''
    if files != []:
        with st.spinner('Processing...'):
            for file in files:
                stringio = StringIO(file.getvalue().decode("utf-8"))
                string_data = stringio.read()
                input_su += string_data + '\n'

            if int(len(extractiveSummarization.arctica(input_su, len(files)))/4) > max_length:
                st.error('Please enter a text in English of maximum {} tokens'.format(max_length))
            else:
                text_summarizer(model, extractiveSummarization.arctica(input_su, len(files)))
```

Figure 3.4 : StringIO usage

— **Transformers** provides general-purpose architectures (BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet, Pegasus. . .) for Natural Language Understanding (NLU) and Natural Language Generation (NLG) with over 32+ pretrained models in 100+ languages and deep interoperability between Jax, PyTorch and TensorFlow.[59]
From this liberary, we have taken two important fuctions : **PegasusForConditionalGeneration** and **PegasusTokenizer**.

— **PegasusTokenizer** : This Method converts characters/sentences into Tokens.

— **PegasusForConditionalGeneration** : The PEGASUS Model with a language modeling head. Can be used for summarization.

— **extractiveSummarization** : This was our made function, we will be detailing its work in the multi-documents part.

```python
def arctica(text, textNumber):

    sentences = tokenize.sent_tokenize(text)

    model = SentenceTransformer('all-mpnet-base-v2')
    for sentence in sentences:

      #Encode all sentences
      embeddings = model.encode(sentence)

      #Compute cosine similarity between all pairs
      cos_sim = util.cos_sim(embeddings, embeddings)

      #Add all pairs to a list with their cosine similarity score
      all_sentence_combinations = []
      for i in range(len(cos_sim)-1):
          for j in range(i+1, len(cos_sim)):
              all_sentence_combinations.append([cos_sim[i][j], i, j])

      #Sort list by the highest cosine similarity score
      all_sentence_combinations = sorted(all_sentence_combinations, key=lambda x: x[0], reverse=True)

      all_sentence = ''
      for score, i, j in all_sentence_combinations[:int(len(sentences)/textNumber)]:
          all_sentence += random.choice([sentences[i], sentences[j]])


      return all_sentence
```

Figure 3.5 : extractiveSummarization corpus

3.2.5.2 User Interface

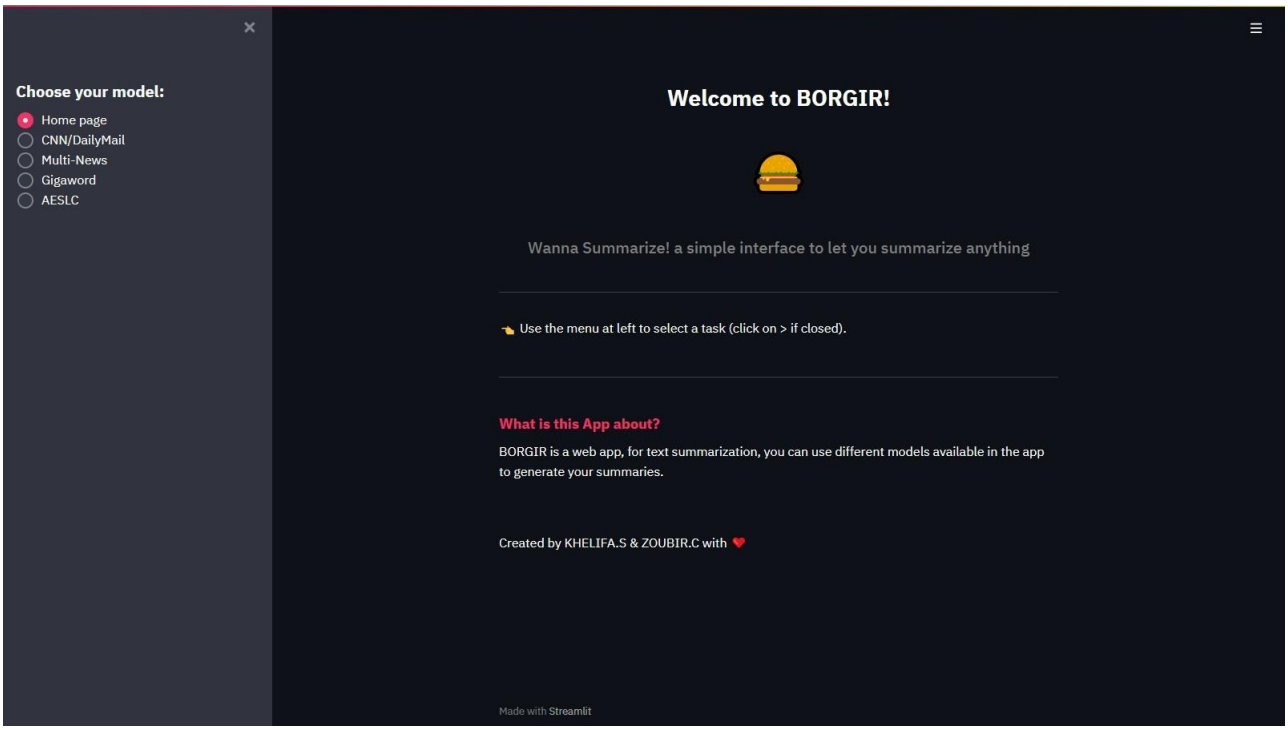

Figure 3.6 : User Interface

We have coded this interface using stearmlit (as said above).

— **I have to put the line here** : was the line that set the title and page icon.

— **I have to put the line here, I have to put the line here, I have to put the line here** : was the line that helped us make the sidebar that is shown on the left of the app.

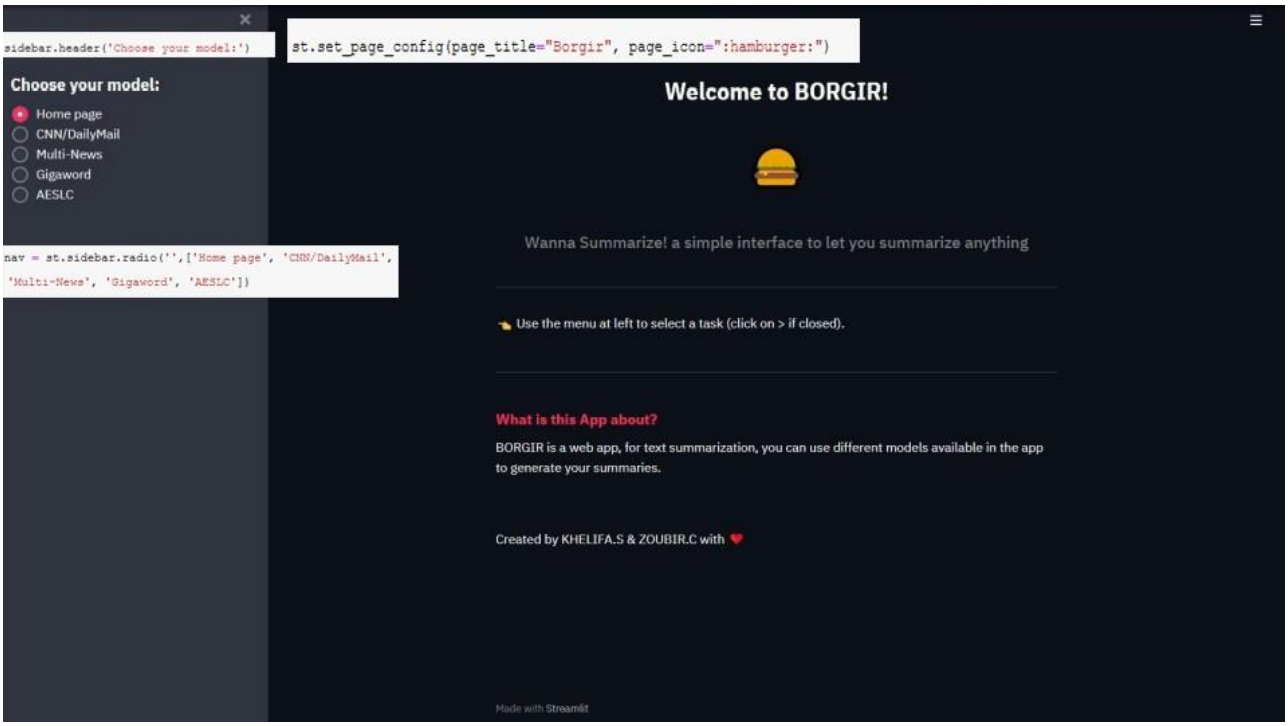

Figure 3.7 : User Interface with code.

3.2.6 Multi-Documents Summarizing

In this section we will be highlighting The technique that helped us build and summarize multi-documents.

We first start with the dependencies, meaning : the libraries needed for our work.

The libraries are the same as the ones used in the mono-document summarazing, we eventually removed the **streamlit** library but we have added the **rouge-score** library that would allow us to callculate the rouge scores and see if the summarizing was pertinent. (we will dive into it in the Experiment chapter.)

```
!pip install transformers #the hugging face library
!pip install sentencepiece #Text Tokenazer
!pip install -U sentence-transformers #sentence embedding library
!pip install rouge-score #ROUGE score library
```

Figure 3.8 : multi-documents dependencies

3.2.6.1 Summarizing Process

We will start by introducing the steps and will explain them one by one

```
[ ]  #Create a dictionary of concatenated texts of a cluster to be summarised
     path_name = '/content/drive/MyDrive/DUC-2004/Cluster_of_Docs/'
     path = os.listdir(path_name)
     general_doc = {}

     for folder_name in path:
       path = os.listdir(path_name + str(folder_name))
       doc_conc = ''

       for file_name in path:
         with open(path_name+ str(folder_name) +'/'+ str(file_name), 'r') as f:
           file_data = f.read()
           doc_conc += file_data
       general_doc[folder_name] = doc_conc
```

```
[ ]  general_doc

     {'d30001t': "   Cambodia's leading opposition party ruled out sharing the presidency of Parliament
      'd30002t': "   Hurricane Mitch paused in its whirl through the western Caribbean on Wednesday to p
      'd30003t': "   A delegation of Chilean legislators lobbying against the possible extradition of Au
      'd30005t': "   MUNICH, Germany (AP) _ U.S. prosecutors have asked for a 20-day extension to provid
      'd30006t': "   The National Basketball Association, embroiled in a labor dispute with its players,
      'd30007t': "   Rebel commanders said Tuesday they were poised to overrun an important government-h
      'd30008t': "   Indonesian President B.J. Habibie finds attending a summit of Asia-Pacific leaders
```

Figure 3.9 : Loading Documents

We start by uploading the documents that are needed to summarize them.
We put each set of documents into a list. these lists contain a multitude of documents that talk about the same topic.
For Example :" **d30001t** will contain documents that talk about political issues. **d30002t** will contains documents that talk about sports. **d30003t** will contain documents that talk about the weather...etc.

— The next step would be to tokenize, or to make the texts that are included into the lists into sentences to facilitate our work.

```
[ ]  #Tokenize the textes into sentences
     general_sent = {}

     for key, value in general_doc.items():
       general_sent[key] = tokenize.sent_tokenize(value)
```

```
general_sent
```

```
{'d30001t': ["   Cambodia's leading opposition party ruled out sharing the presi
  "Disputes over the presidency have been a major stumbling block in talks betwe
  "Cambodian leader Hun Sen, who heads the CPP, has offered to share the legisla
  "The prince's party, in a statement dated Friday and seen Saturday, said such
  '``Co-sharing anything with the CPP means surrendering full power to them.',
  "Furthermore, such a proposal is unconstitutional,'' the faxed statement said.
  "The royalist party also rejected Hun Sen's calls to hold bilateral talks, ins
  "``No party with seats at the National Assembly should be left out if transpar
  'Noting that FUNCINPEC allowed the CPP to hold the presidency despite its win
  "In the most recent elections, held in July, Hun Sen's party collected 64 of t
  'Fearing arrest, many opposition members of Parliament left Cambodia after the
```

Figure 3.10 : Text to Sentence

— We will be moving into the next step, which is **embeddings** and **cosine-similarity**, **putting sentences into pairs** and then **sorting them from high to low cosine-score**.

```
#Encode all sentences
embeddings = model.encode(sentences)

#Compute cosine similarity between all pairs
cos_sim = util.cos_sim(embeddings, embeddings)

#Add all pairs to a list with their cosine similarity score
all_sentence_combinations = []
for i in range(len(cos_sim)-1):
    for j in range(i+1, len(cos_sim)):
        all_sentence_combinations.append([cos_sim[i][j], i, j])

#Sort list by the highest cosine similarity score
all_sentence_combinations = sorted(all_sentence_combinations, key=lambda x: x[0],

all_sentence = ''
#shoose the top high scores in the list
for score, i, j in all_sentence_combinations[:int(len(sentences)/10)]:
    all_sentence += random.choice([sentences[i], sentences[j]])
sent_summaries_dic[sentence] = all_sentence
```

Figure 3.11 : Text to Sentence

— **Embeddings** : turning the said sentences into numbers depending on their semantic values.

```
[ ] embeddings

array([[-0.00559777,  0.0707018 , -0.00820693, ...,  0.01146993,
        -0.016615  , -0.05774559],
       [ 0.01861161,  0.01996453,  0.03047237, ..., -0.00395346,
        -0.02559551, -0.03536284],
       [ 0.02900826,  0.00697633,  0.01594028, ...,  0.00932759,
        -0.04212672, -0.05406781],
       ...,
       [ 0.01745067,  0.00657091,  0.015415  , ..., -0.04585848,
        -0.02869177, -0.05072188],
       [ 0.01988772,  0.10724799, -0.00698555, ...,  0.06279515,
        -0.04122685,  0.00771615],
       [-0.01140975,  0.07620154, -0.00848127, ...,  0.0159965 ,
        -0.06466551,  0.02958634]], dtype=float32)

[ ] len(embeddings[2])

768

[ ] len(embeddings)

412
```

Figure 3.12 : Embedding Score

Cosine-similarity : the cosine-similarity is done by comparing each sentences to the others.

For example : Sentence1 is compared to Sentence2...Sentence3...Sentence4...SentenceN. Sentence2 is compared to Sentence1...Sentence3...Sentence4...SentenceN. The Only sentence that won't be compared is **SentenceN**.

```
[ ] cos_sim

tensor([[1.0000, 0.2941, 0.4788,  ..., 0.2401, 0.0449, 0.0655],
        [0.2941, 1.0000, 0.3295,  ..., 0.2200, 0.0168, 0.0303],
        [0.4788, 0.3295, 1.0000,  ..., 0.2918, 0.1520, 0.0977],
        ...,
        [0.2401, 0.2200, 0.2918,  ..., 1.0000, 0.1974, 0.1483],
        [0.0449, 0.0168, 0.1520,  ..., 0.1974, 1.0000, 0.3184],
        [0.0655, 0.0303, 0.0977,  ..., 0.1483, 0.3184, 1.0000]])
```

Figure 3.13 : cosine-similarity score

— The Last step is take the first **average number of sentences of all documents** of sentence and then create a whole new document with those sentences.

```
[ ] sent_summaries_dic

{'d30001t': "Hun Sen's party won 64 of the 122 seats in parliament in July's national election, but not the two-t
 'd30002t': "``Few houses have remained standing.''``The hurricane has destroyed almost everything,'' said Mike E
 'd30003t': "Pinochet was arrested in London on Oct. 16 at the instigation of Spanish magistrate Baltasar Garzon
 'd30005t': "Bin Laden, a Saudi exile, is thought to be living in Afghanistan.``An important man with close links
 'd30006t': "It doesn't look promising, but we're going to spend the next two days to see if we can give a favora
 'd30007t': "``We have to explain to the world audience who we are and what we're fighting for,'' rebel leader En
 'd30008t': "APEC groups Australia, Brunei, Canada, Chile, China, Hong Kong, Indonesia, Japan, South Korea, Malay
 'd30010t': "``This heroic operation is not the first and not going to be the last,'' the group said in a leaflet
 'd30011t': "He then began speaking out against Mahathir and was arrested Sept. 20.Anwar mounted a nationwide cam
 'd30015t': "One diplomat in Kosovo, who spoke on condition of anonymity, said there has been a significant redep
 'd30017t': "The Korean states were separated into the communist North and the capitalist South in 1945.North Kor
 'd30020t': "Saudi Arabia has in the past said relations would only be normalized when the two cases were solved.
 'd30022t': "He said.Wang was a student leader in the 1989 Tiananmen Square democracy demonstrations.Wang was a s
 'd30024t': "Even as the two leaders spoke, their angry rank-and-file lawmakers were making phone calls trying to
 'd30026t': "Now, the company has to get serious if it is to win the hearts and minds of corporate executives in
 'd30027t': "IMF officials have said they want the Russian government to come up with a sound economic program be
 'd30028t': "Ankara has sent troops to its border with Syria, prompting fears of an attack.Ankara has sent troops
 'd30029t': "A major search involving Australia's navy continued Monday morning for those missing.The nine crew o
 'd30031t': "``We're halfway home,'' Cabana said.``We're halfway home,'' Cabana said.``We're halfway home,'' Caba
 'd30033t': "``The coordinated rate cut could lead to a reduction in the current pessimism and a reduction in fir
 'd30034t': "The half-island territory lies some 2,000 kilometers (1,200 miles) southeast of Jakarta.The half-isl
 'd30036t': "Last year, the prize went to Stanley B. Prusiner of the University of California at San Francisco fo
 'd30037t': "Syria and the previous Israeli government had reached an informal agreement on a full withdrawal fro
 'd30038t': "IOC president Juan Antonio Samaranch immediately disassociated himself from Hodler's allegations.My
 'd30040t': "Since then, Israel has accused the Palestinians of failing to contain anti-Israel violence, and has
 'd30042t': "Annan's comments came Saturday after he and Gadhafi failed to agree on handing over Abdel Basset Ali
 'd30044t': "The Turkish army claims to have all but wiped out the PKK within Turkey but the rebels have havens i
 'd30045t': "If the companies were to merge, it would create the largest U.S. company in terms of revenue.Neither
 'd30046t': "You resign!''Impeach!``I must set the example that I hope President Clinton will follow.'' Livingsto
```

Figure 3.14 : Last Sentences

— Those New documents will be summarized and we will be checking the Rouge scores to check if the summary was pertinent.

```
[ ]  final_summaries = {}

     for model in models:
       final_summaries[model] = {}
       for key, value in sent_summaries_dic.items():
         final_summaries[model][key] = text_summarizer(model, value)
```


```
[ ]  orderd_target_summarie_dic = collections.OrderedDict(sorted(target_summarie_dic.items()))
     score_list = {}

     scorer = rouge_scorer.RougeScorer(['rouge1','rouge2', 'rougeL'], use_stemmer=True)

     for model in models:
       orederd_final_summaries = collections.OrderedDict(sorted(final_summaries[model].items()))
       score_list[model] = {}
       score_list[model]['rouge_1'] = []
       score_list[model]['rouge_2'] = []
       score_list[model]['rouge_L'] = []
```

Figure 3.15 : Summary and using rouge-score

— The Rouge score will be discussed in the next chapter.

3.3 Conclusion

In this chapter, we have covered everything about the steps needed to build and create a good summarizing model.
We have briefly described each step and proposed methods that were used during preprocessing in the generation of multi-document summaries.
In the next and last chapter, we will be experimenting every detail written on the previous chapters. We will also compare our work with those of last year.

Chapter 4

Experiments

4.1 Introduction

In this final chapter, we will aboard our Application : Borgir, and experience its pertinence by comparing our work with last year's work.

4.2 Environment

4.2.1 Google Colab

Colab is a free notebook environment that runs entirely in the cloud. It lets you and your team members edit documents, the way you work with Google Docs. Colab supports many **popular machine learning libraries** which can be easily loaded in notebook. Here are some of the power of Google Colab :

— **CPU :** Intel(R) Xeon(R) CPU @ 2.30GHz.

— **GPU :** Tesla P100-PCIE-16GB.

— **Disk** :114GB.

— **RAM** :26G.

— **Python Version** :3.9.7. Google Colab was helpful for the pre-training and showing the score of ROUGE.

4.2.2 Personal Environment

— **CPU :** Processor Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz, 2712 Mhz, 4 Core(s), 8 Logical Processor(s).

— **Operating-System** : Microsoft Windows 10 Pro.

— **RAM :** 32.0 GB.

— **GPU :** NVIDIA Quadro M2000M.

— **Disk :** SSD 500GB.

— **Python Version :** 3.9.7.

4.2.3 Programming Language and IDE

4.2.3.1 Python

We used Python in both Backend and FrontEnd.
Python is a high level programming language that is capable of doing almost anything, it is very helpful in terms of solving Big Problems because of its basic syntax.
Python has a massive library set especially in machine learning but that doesn't stop it from being able to build websites and desktop applications.

Figure 4.1 : Logo of Python

4.2.3.2 Integrated Development Environment

— **Jupyter Note-Book** : Jupyter Notebook is an open-source web-based IDE that allows creating documents containing code, images, narrative text heavily oriented to Machine learning.

Figure 4.2 : Jupyter Logo

— **Spyder** : is a free and open source scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts.

Figure 4.3 : Spyder Logo

4.2.4 Used Libraries

4.2.4.1 Transformers

Transformers provides general-purpose architectures (BERT, GPT-3,Pegasus for Natural Language Understanding (NLU) and Natural Language Generation (NLG) with over 32+ pretrained models in 100+ languages and deep interoperability between Jax, PyTorch and TensorFlow.

4.2.4.2 StreamLit

is an open-source app framework for Machine Learning and Data Science teams. Create beautiful data apps in hours, not weeks. All in pure Python.

4.2.4.3 IO

The io module provides Python's main facilities for dealing with various types of I/O.

4.2.4.4 RougeScore

A native python implementation of ROUGE, designed to replicate results from the original perl package.

4.2.5 DataSets

4.2.5.1 AESLEC

It was created by Zhang and Tetreault, AESLC consists of 18,000 email bodies and their subjects from the Enron Corpus, a collection of email messages of employees in the Enron corporation, designed for abstractive email summarization and available for public usage.

4.2.5.2 CNN-DailyMail

This DataSet was taking from the best summrzation exprets who are : Journalists. CNN/DailyMain represtents two datasets created by Hermann et al. Containing over 300,000 articles in total (93,000 for CNN and 220,000 from Dailymail newspaper). This dataset has each article paired with a short set of summarized bullets points that represent meaningful highlights of the piece.

4.2.5.3 GigaWord

contains 4M examples extracted from news articles (seven publishers) from the Giga-word corpus. The task is to generate the headline from the first sentence. [31]

4.2.5.4 Multi-News

is a multi-document summarization dataset consisting of 56k pairs of news articles and their human-written summaries from the site newser.com [32]

4.2.6 Evaluation System

The Evaluation System that was used is the well known ROUGE. ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. ROUGE works by comparing the generated summaries against one or a set of other summaries usually human-written by human-written ones and is recall-oriented. By comparing overlapping words in n-grames between two texts an calculating the precision and the F-measure.

— **Recall :** The recall is the quantity of right information recovered by a system compared to what it should recover.

$$Recall=\frac{\text{number\_of\_overlapping\_words}}{\text{total\_words\_in\_reference\_summary}}$$

— **Precision** : The precision is the quantity of right information collected by a system compared to what it has recovered.

$$Precision=\frac{\text{number\_of\_overlapping\_words}}{\text{total\_words\_in\_system\_summary}}$$

— **F-Measure** : The F-measure is a mix between the recall and the precision that balances both matters in one number. F-measure is calculated as :

$$F-Measure=\frac{(1+\beta\,2\,)\mathrm{P}*\mathrm{R}}{\beta\,2\,\mathrm{P}+\mathrm{R}}$$

— **ROUGE-N** is a recall of n-grams between the candidate summary and a set of reference summaries.

$$RougeN=\frac{\sum_{S\epsilon Summref}\sum_{N-gram\epsilon S}\text{Count match (N}-\text{gram)}}{\sum_{S\epsilon Summref}\sum_{N-gram\epsilon S}\text{Count(N}-\text{gram)}}$$

— **ROUGE-L**To overcome the weakness of the ROUGE-N, ROUGE-L employs the concept of the longest common subsequences (LCS).

$$RougeLCS = \frac{\sum_{i=1}^{u} \text{LCS U (s i ,C)}}{m} \text{ , } PLCS = \frac{\sum_{i=1}^{u} \text{LCS U (s i ,C)}}{n}$$

4.3 Experiments

4.3.1 Mono-Documents : BORGIR

All those chapters to finalize our application. Borgir, the app that can summarize anything. Our Application have shown incredible results in regards of mono-documents in terms of summerzation. We have tested out some texts using the different datasets.
We will be showing some highlights, to show you how it works.

— First, we will be starting with the Main page,

— On the left side of the App you can find the different dataset that you want to use.

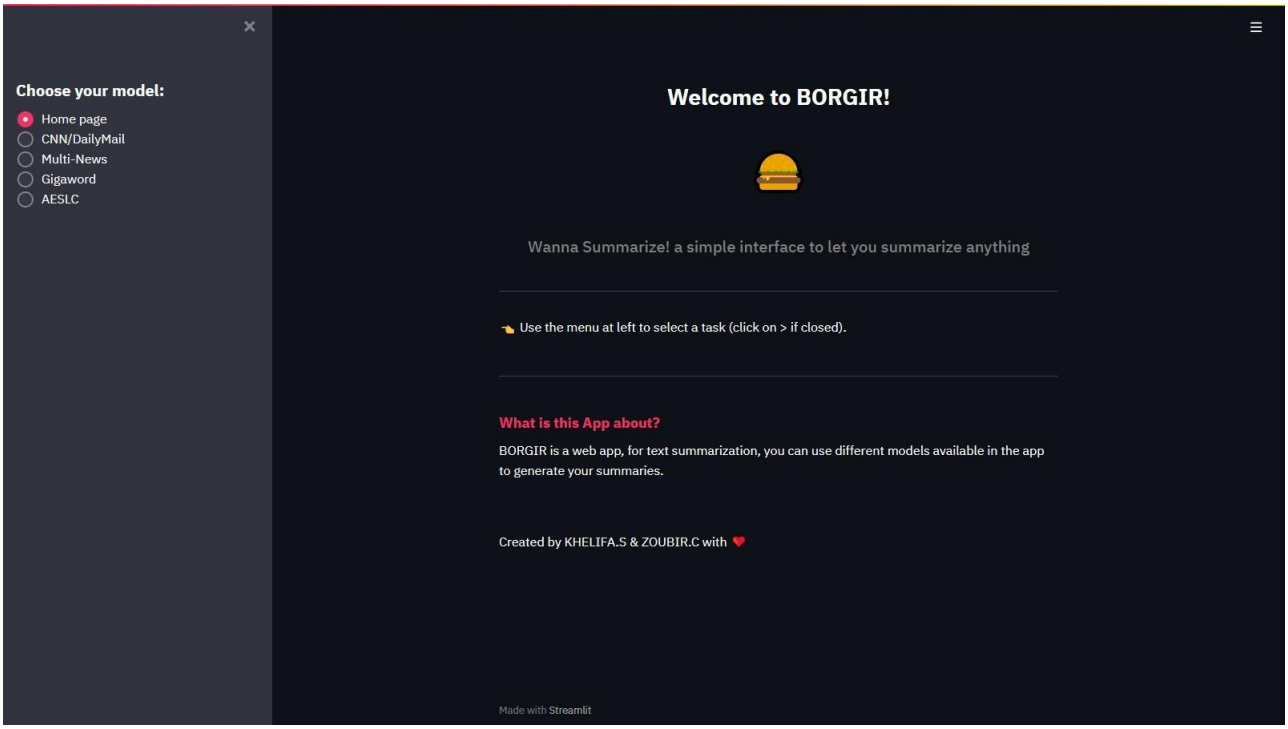

Figure 4.4 Main Page of App

Like any Good App, Borgir offers the choice of, whether you want to copy paste a text or upload a file.

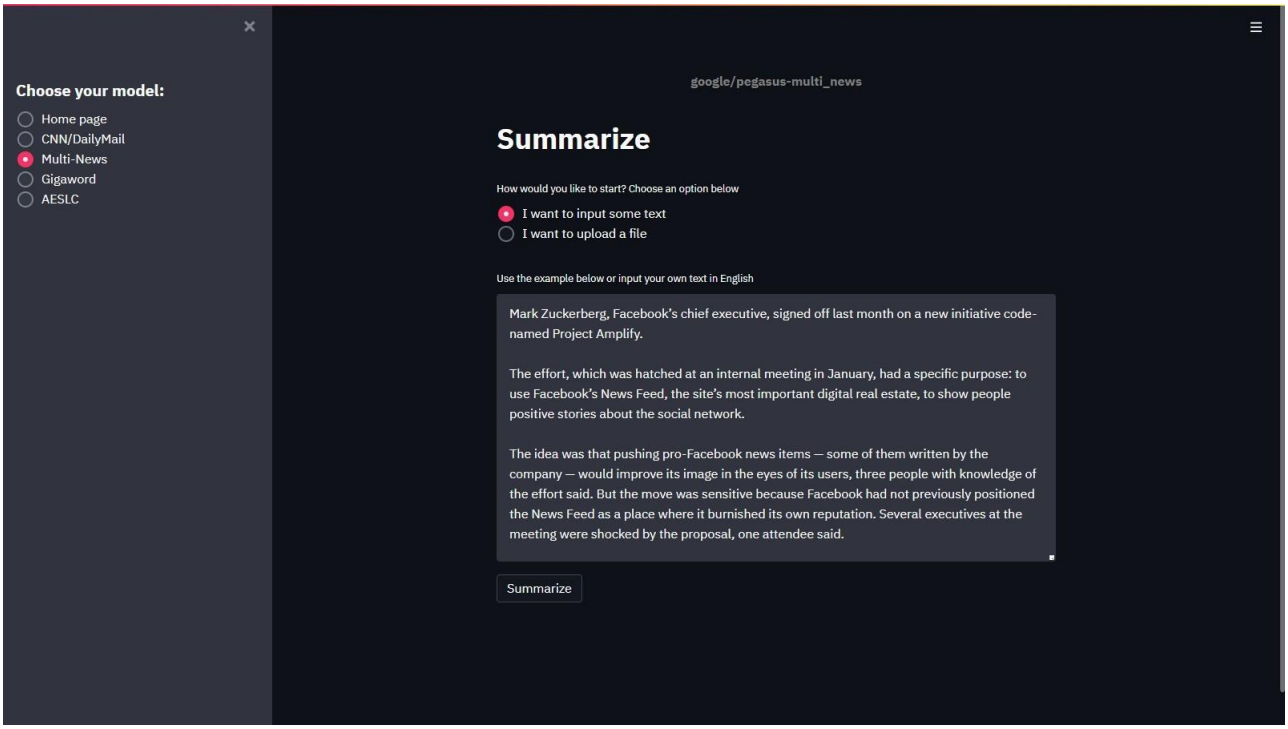

Figure 4.5 multiple choices

When typing into the "upload a file" the box where you can write a text disappears and gives place of a command that you click on to upload your file.

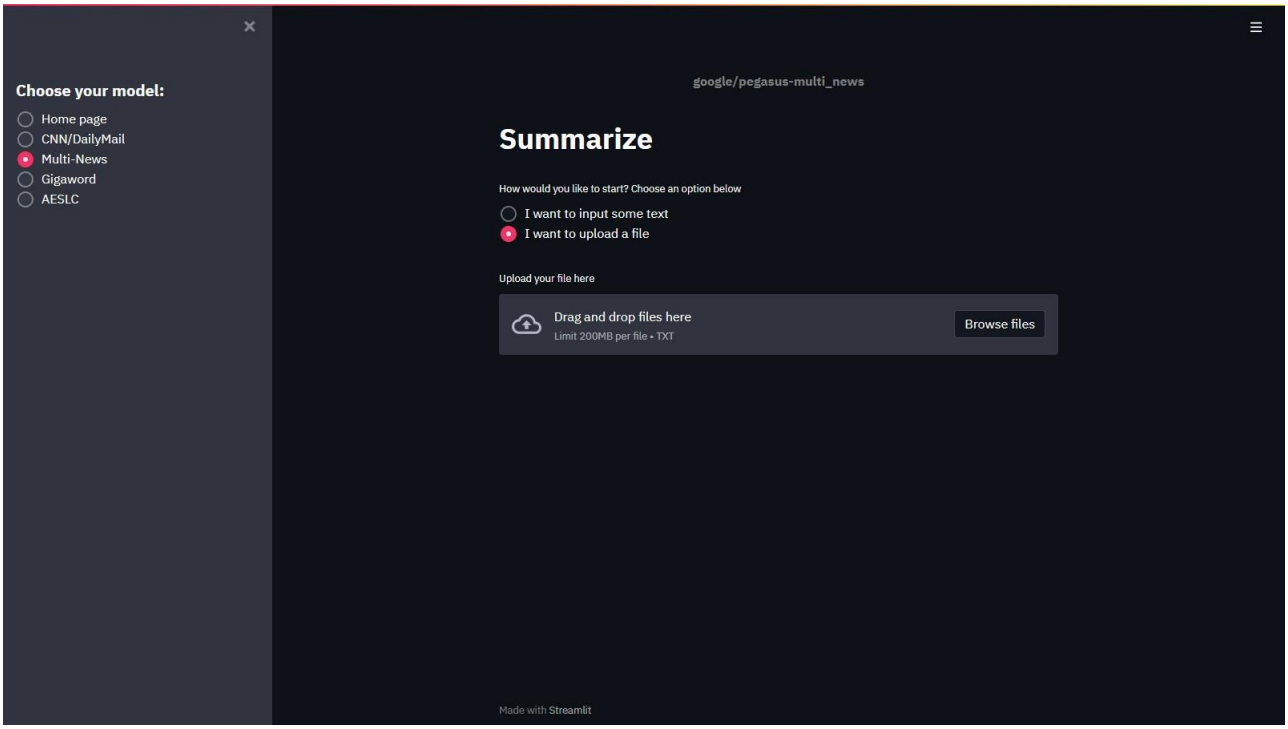

Figure 4.6 Upload File

Borgir makes sure that your text is in **english** and doesn't surpass the capacity of **511**

tokens. (tokens are approximately 4 characters.)

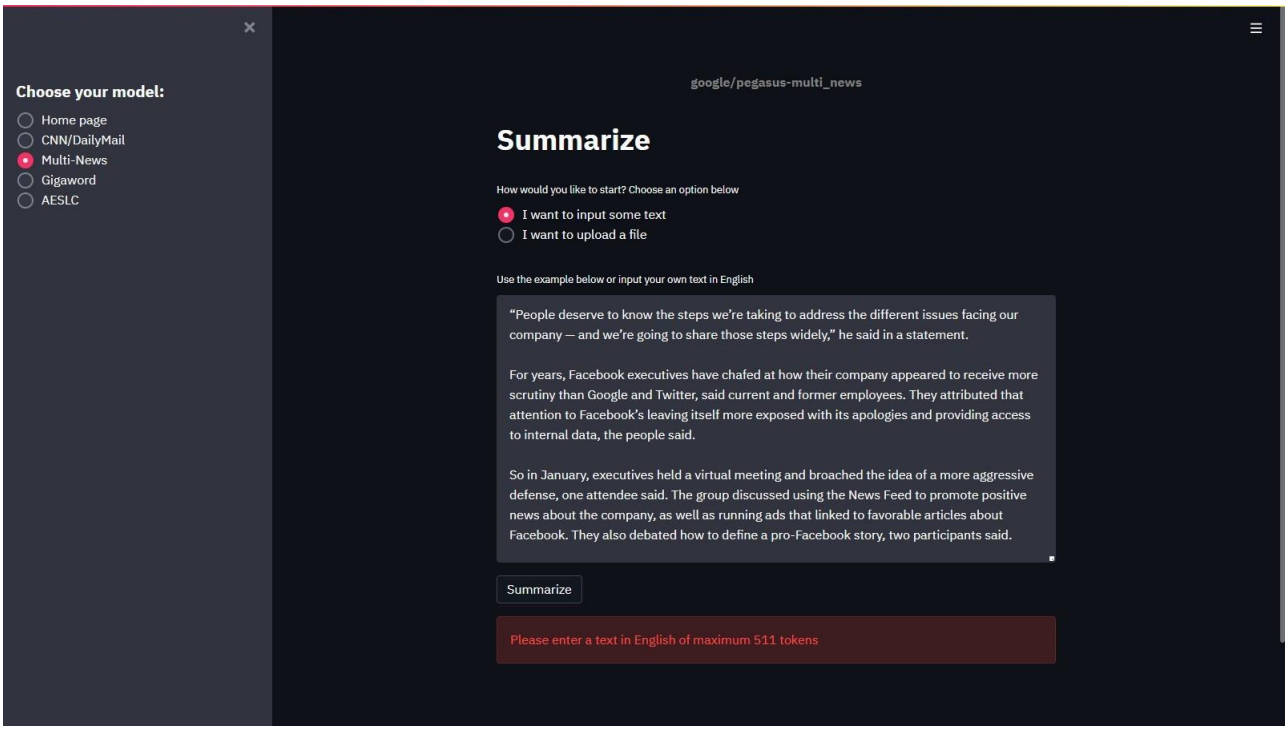

Figure 4.7 Error in App

Once you choose the dataset you want to use, and type the text you want to summarize, the application shows you a beautiful dispell of output with your summarized text.

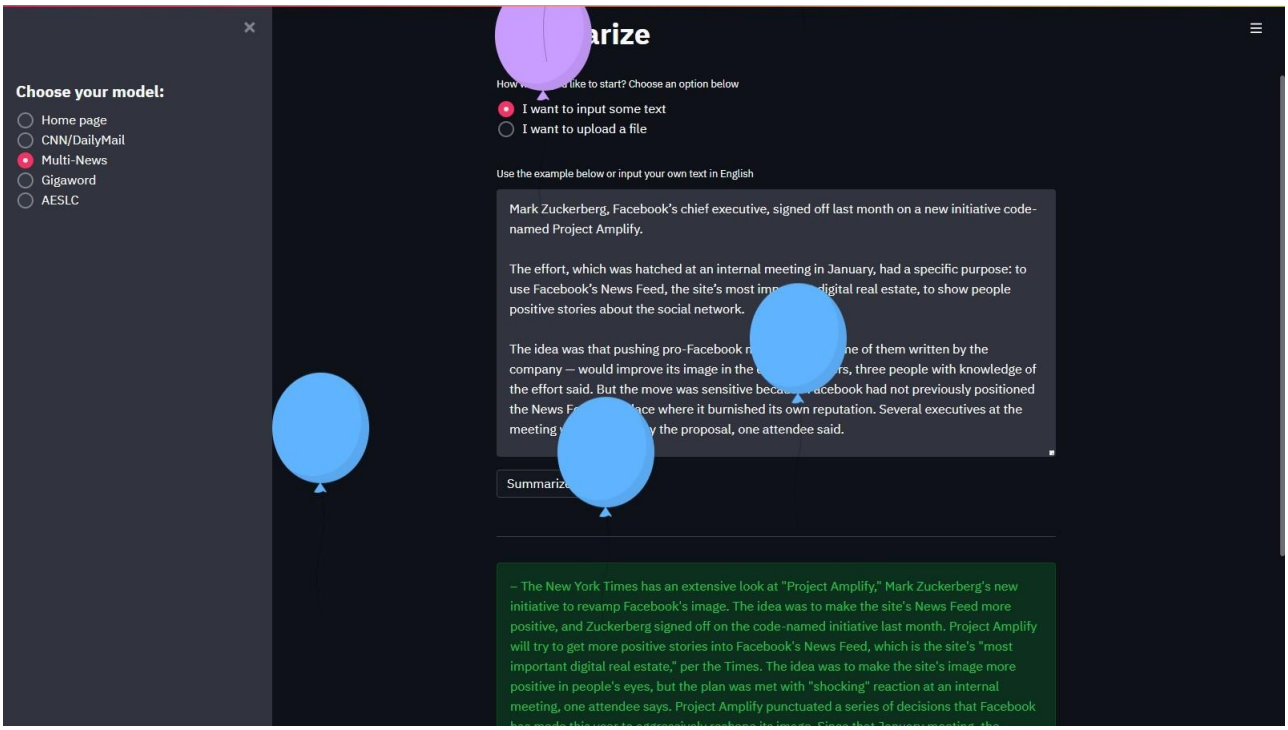

Figure 4.8 Result of Summary

4.4 Multi-Documents summarization

Now that it is all settled, we will be trying to evaluate and compare our work with others work, especially our comrades from the previous year.
Our comrades worked on a Multi-Documents approach that consisted of :

— feeding the models the documents concatenated with each other, then observe the outputs and compare them with the reference summaries.

— pre-processing the input as in the extractive multi-document summarization approach, once with similarity threshold >= 0.8 and another one with threshold >= 0.9, and comparing them with the reference summaries. Here were their results :

— The result of the first approach (feeding models) :

	Rouge 1	Rouge 2	Rouge L
DistilBart-CG	7.11/23.04/10.83	0.66/2.13/1.01	5.07/16.44/7.72
DistilBart-CA	14.36/39.42/20.96	2.89/8.17/4.25	9.09/25.20/13.30
DistilBart-MC	29.96/31.26/29.94	5.96/6.34/5.98	15.88/16.79/15.93
CNN/DailyMail	**18.20/39.41/24.75**	**3.83/8.38/5.22**	**10.45/22.73/14.24**

Table 4.1 Previous ROUGE Results (Concatenated Documents)

— The result of the second approach (Pre-processing with similarity threshold >=0.8)

	Rouge 1	Rouge 2	Rouge L
DistilBart-CG	7.09/23.12/10.81	0.61/2.01/0.94	4.91/16.05/7.49
DistilBart-CA	15.13/39.80/21.87	3.18/8.50/4.62	9.66/25.57/13.99
DistilBart-MC	30.77/30.66/30.27	6.10/5.97/5.96	16.30/16.44/16.11
CNN/DailyMail	**20.02/40.34/26.53**	**4.44/9.89/5.89**	**11.16/22.62/14.82**

Table 4.2 Previous ROUGE Results (Threshold Similarity >=0.8)

— Another results of the second approach (Pre-processing with similarity threshold >=0.9)

	Rouge 1	Rouge 2	Rouge L
DistilBart-CG	7.52/24.30/11.46	0.72/2.31/1.09	5.23/16.98/7.97
DistilBart-CA	18.30/44.47/25.87	4.26/10.45/6.04	10.90/26.60/15.43
DistilBart-MC	32.69/35.85/33.48	6.96/7.75/7.71	16.94/18.86/17.45
CNN/DailyMail	**20.55/42.21/27.47**	**4.68/9.59/6.25**	**11.66/24.23/15.65**

Table 4.3 Previous ROUGE Results (Threshold Similarity >=0.9)

To make our experiments even stronger, we used one of the last year's method which was pre-processing the input as in the extractive multi-document summarizing approach, with similarity threshold >= 0.7, similarity threshold >= 0.8, similarity threshold >= 0.9

	Rouge 1	Rouge 2	Rouge L
AESLC	2.83/40.41/4.93	0.35/9.25/0.62	2.06/34.93/3.65
CNN/DailyMail	**21.46/38.21/27.15**	**3.76/6.62/4.74**	**11.94/21.44/15.14**
GigaWord	5.69/29.89/9.53	0.65/3.44/1.09	4.13/22.23/6.93
Multi-News	39.85/29.55/32.89	8.11/6.0/6.65	20.6/15.74/17.26

Table 4.4 Rouge Results using threshold >=0.7

We continued and tried to optimize our work to its maximum and tried a similarity threshold >=0.8

	Rouge 1	Rouge 2	Rouge L
AESLC	3.12/42.62/5.47	0.28/8.44/0.51	2.32/35.92/4.1
CNN/DailyMail	**18.79/37.47/24.49**	**2.97/5.94/3.87**	**10.58/21.64/13.89**
GigaWord	6.12/32.1/10.23	0.78/4.32/1.31	4.58/24.4/7.67
Multi-News	37.6/29.81/32.28	7.18/5.54/6.09	20.34/16.34/17.59

Table 4.5 Rouge Results using threshold >=0.8

This Method showed better results by :

CNN-Daily Mail of last year 0.8 :

— ROUGE1 : 20.02/40.34/26.53

— ROUGE2 : 4.44/9.89/5.89

— ROUGE-L : 11.16/22.62/14.82

CNN-Daily Mail of This year 0.8 :

— Rouge1 : 19.03/38.48/24.95

— Rouge 2 : 4.87/9.46/6.33

— ROUGE-L : 11.43/23.32/15.03

	Rouge 1	Rouge 2	Rouge L
AESLC	2.96/43.91/5.23	0.66/17.41/1.2	2.29/38.52/4.08
CNN/DailyMail	**16.5/38.34/22.45**	**3.38/8.42/4.69**	**10.22/24.35/14.04**
GigaWord	5.96/30.7/9.96	0.73/3.8/1.22	4.31/22.4/7.2
Multi-News	37.32/28.48/31.42	7.47/5.24/5.99	19.65/15.07/16.57

Table 4.6 Rouge Results using threshold >=0.9

CNN-Daily Mail of last year 0.9 :

— ROUGE1 : 20.55/42.21/27.47

— ROUGE2 : 4.69/9.59/6.25

— ROUGE-L : 11.66/24.23/15.65

CNN-Daily Mail of This year 0.9 :

— Rouge1 : 16.5/38.34/22.45

— Rouge 2 : 3.38/8.42/4.69

— ROUGE-L : 10.22/24.35/14.04

We noticed something when we were experimenting the threshold method : Whenever we increased the threshold similarity, the ROUGE score of our summarizing was decreasing. To confirm our hypnotises we tried a threshold >= 0.95, and here were the results :

	Rouge 1	Rouge 2	Rouge L
AESLC	2.46/35.45/4.31	0.47/7.51/0.84	1.86/30.58/3.29
CNN/DailyMail	**15.5/35.93/20.75**	**2.79/5.99/3.61**	**9.45/23.62/12.87**
GigaWord	4.7/25.83/7.92	0.43/2.52/0.73	3.32/18.48/5.59
Multi-News	31.79/29.16/29.38	5.56/4.67/4.94	16.83/15.68/15.65

Table 4.7 Rouge Results using threshold >=0.95

CNN-Daily Mail of This year :

- Rouge1 : 15.5/35.93/20.75
- Rouge 2 : 2.79/5.99/3.61
- ROUGE-L : 9.45/23.62/12.87

This phenomena is due to the lack of comparison sentences. Which means : whenever we get closer to one, the similarity is almost null.

We performed another method which consisted of :

- Concatenating ten documents into one document.
- Getting the sentences from that documents and compare each sentence with all others.
- Acquire the top average document length in the cluster. (the top average comes in pair.)
- choose one sentence randomly from the pair of sentence score.
- Concatenate the chosen sentences into one document.
- Summarize the new document. And here were the results :

	Rouge 1	Rouge 2	Rouge L
AESLC	2.93/40.18/5.11	0.48/5.95/0.82	2.33/34.7/4.09
CNN/DailyMail	**21.06/40.44/27.26**	**4.55/9.25/6.01**	**12.38/24.28/16.13**
GigaWord	6.62/34.13/11.01	0.97/5.24/1.62	4.55/23.3/7.56
Multi-News	40.38/31.67/34.5	9.07/6.62/7.46	20.7/16.25/17.69

Table 4.8 Rouge Results using Concatenation method

Our dataset gave better results than the previous works. Example : CNN-Daily Mail of last year :

- ROUGE1 : 18.12/39.41/29.94
- ROUGE2 : 3.83/8.38/5.22
- ROUGE-L : 10.45/22.73/14.24

CNN-Daily Mail of This year :

- Rouge1 : 19.71/40.85/26.32
- Rouge 2 : 4.51/7.94/5.27
- ROUGE-L : 12.22/25.28/16.31

This method had shown flourishing results compared to the threshold method comparing to the other threshold results.

Adding a few steps was helpful for us to reach the perfect text summarizing.

4.4.1 Struggles

: We have faced **two obvious struggles** :

— The first one was TOO well trained, when we put a text into our summerziation app, the app used to summerazie it into only one sentence. For Example : We gave a Wikipedia description of the country **Canada**, the app summarized into only one sentence : **Canada is a Country.**

— The Second struggle was misunderstanding of the topic : Our models couldn't understand or misunderstood the main topic of our speech and usually went out of context. For example :
We put a text about **a Presidential meeting** that started with :" The Queen of United Kingdom has met Obama..."
The model understood that we were talking about the United Kingdom and went on describing the country...

4.4.2 Discussion

Our work consisted in fine-tuning a **pegasus** model to summarize several documents and compare our work to all the works related with that field.

We successfully built and designed an application that summarized documents whether mono or multi-documents.

By using ROUGE scores, we have scored higher results than the previous works, a thing which is promising.

Updates will be made in this field as the evolution of technology will evolve.

4.5 Conclusion

This chapter described the technical side of our project including its software and hardware information. The actual results were presented with an in-depth analysis and explanation. We compared our models with a baseline model. Since the ROUGE metric has many weaknesses and does not provide 100 percent reliable scores for this specific task, we need an evaluation metric dedicated to abstractive summarization. Despite that, we are positive that transformer-based models have great potential and can be further improved.

General Conclusion

In this work, our objective was to propose a solution to the problem of multi-document abstractive summarization. Automatic text summarization methods are essential as the amount of available data and the need of relevant, fast, and concise information is continuously increasing. Our choice leaned towards the machine learning approach, which has proven its effectiveness in this area. Thus, we proposed to finetune an existing model called Pegasus. We experimented with different datasets : Gigaword, Multi-news, AESLC and we found that the best results were obtained when the model is finetuned on CNN/Daily-Mail. We also attempted to find a method that improves the quality of our generated summaries and proposed a solution based on replacing similar sentences with one that has the same meaning, and that is a part of their cluster, as our algorithm groups sentences into clusters. This solution was first used with an extractive model, then on our abstractive resulted models and we found that all of them achieved better performance compared to last year's work. The main contributions of our work are :

— We fine-tuned an existing model on new datasets for single document summarization, then we adapted the resulting models for generating abstractive summaries of multiple documents.

— We proposed an algorithm to adapt our models to generate a summary for multiple documents for both the extractive and abstractive type.

The principal issue we faced during this project is related to the resources needed for the execution. Google Colaboratory resources are not unlimited, although it is free to use, the GPU provided is not powerful enough for big models to train fast. Also, an instance connection limit is around 10 hours then it disconnects; so when our execution was not complete and saved, we lost multiple times hours of progress. Besides, the GPU usage is also limited, after exceeding the limit one has to wait more than 8 hours to be able to use it again. To overcome this particular issue, we had to create multiple Google accounts then save the resulted model each time and execute it on another account. Finally, the biggest

problem is the limit of RAM available since users can use up to 12GB of memory.

As a perspective, we would like in the future to :

— Complete the fine-tuning on Multi-News.

— Fine-tune the model on the paid version of Gigaword.

— Propose a post-processing method to clean the text by eliminating the last sentence if it returns no information or completes the sentence using a trained model that takes the summary and non summarized document as input.

— Try extractive summarization followed by abstractive summarization, where in the extractive step, we would like to choose the top k sentences without exceeding the model maximum length.

— Try to apply successive abstractive summarization either by splitting the document into shorter parts, summarize each part and concatenate the respective summaries with post-processing ; or summarize a part of the document then concatenate the resulted summary with another part of the document, and so on until we would get the final summary. As expansive as it looks, we think it may give good results.

Bibliography

[1] Charu C. Aggarwal, Neural Networks and Deep Learning: A Textbook, September 2018

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville, “Deep Learning”, October 2015

[3] Aurélien Géron, Hands on Machine Learning Deep Learning and Sckitlearn, 13 mars 2017

[4] Umberto Michelucci, Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection, September 28, 2019.

[5] Simeon Kostadinov, Recurrent Neural Networks with Python Quick Start Guide: Sequential Learning and Language Modeling with TensorFlow Simeon Kostadinov, November 29, 2018.

[6] Vladimir Bok, Jakub Langr, GANs in Action: Deep Learning with Generative Adversarial Networks, September 9, 2019.

[7] Leonardo De Marchi, Hands-On Neural Networks: Learn how to Build and Train Your First Neural Network Model Using Python Leonardo De Marchi, May 30, 2019.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit; Llion Jones, Aidan N. Gomez, “ Attention is all you need” 12 June 2017 https://arxiv.org/abs/1706.0376

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Google AI, 2018

[10] Jay Alammar, The Illustrated Transformer, June 27, 2018, https://jalammar.github.io/illustrated-transformer/

[11] Jingqing Zhang, Yao Zhao, Mohammad Saleh; Peter J. Liu, PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization, Google AI https://arxiv.org/pdf/1912.08777.pdf

[12] Expert Nicola Taccone, What is summary? Definition and concept https://study.com/academy/lesson/what-is-a-summary-definition-lesson- quiz.html

[13] Luis Gonçalves. Automatic Text Summarization with Machine Learning — An overview. Apr. 2020. url : https://medium.com/luisfredgs/automatic-text-summarization-with-machine-learning-an-overview-68ded5717a25.

[14] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, et al. Text Summarization Techniques: A Brief

Survey. 2017. eprint: arXiv:1707.02268.

[15] Roshna Chettri and Udit Kr. Chakraborty. “Automatic Text Summarization”. In: 2017.

[16] Hamza Shabbir Moiyadi, Harsh Desai, Dhairya Pawar, et al. “NLP Based Text Summarization Using Semantic Analysis”. In: International Journal of Advanced Engineering, Management and Science 2.10 (Oct. 2016).

[17] P. B. Baxendale. “Machine-Made Index for Technical Literature: An Experi- ment”. In: IBM J. Res. Dev. 2.4 (Oct. 1958), pp. 354–361. issn: 0018-8646. doi:10.1147/rd.24.0354. url:https://doi.org/10.1147/rd.24.0354.

[18] G. Salton and C. S. Yang. “On the specification of term values in automaticindexing.” In: Journal of Documentation. 29.4 (1973), pp. 351–372.

[19] H. P. Edmundson. “New Methods in Automatic Extracting”. In:J. ACM 16.2(Apr.1969), pp. 264–285. issn: 0004-5411. doi: 10.1145/321510.321519. url: https://doi.org/10.1145/321510.321519.

[20] Mohamed Abdel Fattah and Fuji Ren. “GA, MR, FFNN, PNN and GMM Based Models for Automatic Text Summarization”. In: Comput. Speech Lang. 23.1 Hongyan Jing. “Sentence Reduction for Automatic Text Summarization”. In: Proceedings of the Sixth Conference on Applied Natural Language Processing.ANLC ’00. Seattle, Washington: Association for Computational Linguistics,2000, pp. 310–315. doi: 10.3115/974147.974190. url: https://doi.org/10.3115/974147.974190.

[21] Gerard Salton, Amit Singhal, Mandar Mitra, et al. “Automatic text structuring and summarization”. In: Information Processing and Management 33.2 (1997).Methods and Tools for the Automatic Construction of Hypertext, pp. 193–207. issn: 0306-4573. doi: https://doi.org/10.1016/ S0306-4573(96) 00062- 3. url: http://www.sciencedirect.com/science/article/pii/S0306457396000623.

[22] Khushboo Thakkar, Rajiv Dharaskar, and Manoj Chandak. “Graph-Based Al- gorithms for Text Summarization”. In: Emerging Trends in Engineering andTechnology, International Conference on 0 (Nov. 2010), pp. 516–519. doi: 10.1109/ICETET.2010.104.

[23] Kevin Knight and Daniel Marcu. “Summarization beyond sentence extraction: A probabilistic approach to sentence compression”. In: Artificial Intelligence 139(July 2002), pp.91–107. doi: 10.1016/S0004-3702(02)00222-9.

[24] Hongyan Jing and Kathleen R. McKeown. “The Decomposition of Human- Written Summary Sentences”. In: Proceedings of the 22nd Annual InternationalACM SIGIR Conference on Research and Development in Information Re trieval. SIGIR ’99. Berkeley, California, USA: Association for Computing Ma-chinery, 1999, pp. 129–136. isBn: 1581130961. doi: 10.1145/312624.312666. url: https://doi.org/10.1145/312624.312666.

[25] Su Jeong Choi, Ian Jung, Seyoung Park, et al. “Abstractive Sentence Compres-sion with Event Attention”. In: 2019.

V. A. Yatsko, M. S. Starikov, and A. V. Butakov. “Automatic Genre Recognition and Adaptive Text Summarization”. In: Autom. Doc. Math. Linguist. 44.3 (June2010), pp. 111–120. issn: 0005-1055. doi: 10.3103/S0005105510030027. url:https://doi.org/10.3103/S0005105510030027.

[26] Kam-Fai Wong, Mingli Wu, and Wenjie Li. “Extractive Summarization Using Supervised and Semi-Supervised Learning”. In: Proceedings of the 22nd Inter-national Conference on Computational

Linguistics (Coling 2008). Manchester, UK: Coling 2008 Organizing Committee, Aug. 2008, pp. 985–992. url: https:
//www.aclweb.org/anthology/C08-1124.

[27] Yang Liu. "Fine-tune BERT for Extractive Summarization". In: ArXiv abs/1903.10318(2019).

[28] Alexander M. Rush, Sumit Chopra, and Jason Weston. "A Neural Attention Model for Abstractive Sentence Summarization". In: Proceedings of the 2015Conference on Empirical Methods in Natural Language Processing. Lisbon, Por-tugal: Association for Computational Linguistics, Sept. 2015, pp. 379–389. doi:
10.18653/v1/D15-1044. url: https://www.aclweb.org/anthology/D15-1044.

[29] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, et al. "Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond". In: Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learn ing. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 280–290. doi: 10.18653/v1/K16-1028. url: https://www.aclweb.org/anthology/K16-1028.

[30] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. "Neural Machine Transla- tion by Jointly Learning to Align and Translate". In: ArXiv 1409 (Sept. 2014).

[31] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, et al. "Empirical Evalu- ation of Gated Recurrent Neural Networks on Sequence Modeling". In: ArXivabs/1412.3555 (2014).

[32] Alexander M. Rush, Sumit Chopra, and Jason Weston. "A Neural Attention Model for Abstractive Sentence Summarization". In: Proceedings of the 2015Conference on Empirical Methodsin Natural Language Processing. Lisbon, Por-tugal: Association for Computational Linguistics, Sept. 2015, pp. 379–389. doi:
10.18653/v1/D15-1044. url: https://www.aclweb.org/anthology/D15-1044.

[33] Thomas Roellke, Information Retriveal Models, Foundations and Relationships, Morgan and ClaypoolPublishers; 07/13/2013A

[34] H. P. Luhn. "A Business Intelligence System". In: IBM J. Res. Dev. 2.4 (Oct. 1958), pp. 314–319. issn: 0018-8646. doi : 10.1147/rd.24.0314. url : https://doi.org/10.1147/rd.24.0314.

[35] Chin-Yew Lin and Eduard Hovy, From Single to Multi-document Summarization:A Prototype System and its Evaluation, University of Southern California / Information Sciences Institute

[36] Jingqing Zhang,Yao Zhao Mohammad Saleh Peter J. Liu, PEGASUS: Pre-training with ExtractedGap-sentences for Abstractive Summarization, https://arxiv.org/pdf/1912.08777.pdf

[37] Julian Kupiec, Jan Pedersen, and Francine Chen. "A Trainable Document Sum- marizer". In: Proceedings of the 18th Annual International ACM SIGIR Confer-ence on Research and Development in Information Retrieval. SIGIR '95. Seattle, Washington, USA:Association for Computing Machinery, 1995, pp. 68–73. isbn :0897917146. doi : 10.1145/215206.215333. url : https://doi.org/10.1145/215206.215333.

[38] Eibe Frank, Mark A. Hall Christopher Pal; Data Mining: Practical Machine Learning Tools andTechniques (Morgan Kaufmann Series in Data Management Systems); 2003.

[39] Science4all, https://www.youtube.com/watch?v=JikwRMXJzk0&t=850s (September 4th 2021)

[40] The A.I Hacker, Micheal Phi https://www.youtube.com/watch?v=4Bdc55j80l8 (Sept10th2021)

[41] https://www.geeksforgeeks.org/stringio-module-in-python/

[42] https://huggingface.co/transformers/model_doc/pegasus.html