POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Politecnico
di Torino

Master's Degree Thesis

Transformers-based Abstractive Summarization for the Generation of Patent Claims

Supervisors

Prof. LUCA CAGLIERO

Prof. MORENO LA QUATRA

Candidate

SARA MORENO

April 2023

Summary

A patent is a kind of intellectual property that grants its owner the authority to exclude others to make, sell, or use an invention for a fixed amount of time in exchange for sufficient disclosure of the invention.

Each patent is made of five main parts: abstract, background, summary, description and claims. If the patent is accurately written, only the claims formulation task is up to the Intellectual Property (IP) attorney.
The most important claim is the first one: it is composed of a single sentence, has to set out the distinctive features of the invention and underline the differences from the inventions already present in the same or similar field.

Since the patent filing rate is constantly increasing it is necessary to find ways to fasten the analyses of patents in order to keep up with the innovations.

In this thesis, the goal is to generate the first claim of a patent document using abstractive summarization techniques that can understand the context and meaning of the input text, and generate fluent and coherent first claims: this is done because the IP attorney task can be thought of as a summarization task.

The study focuses on two main research questions: which patent sections are the most effective for generating the first claim, and how does the length of the input text impact the performance of the summarization models.

This research could have significant implications for the legal and innovation communities by improving the accuracy and efficiency of automated patent claim generation.

Seven different input texts are analyzed, including single sections as well as combinations of two sections. The boundaries of each section are highlighted using special tokens to help the models recognize the semantic content.

To investigate the impact of context width, two models are compared: PEGASUS and BigBird-PEGASUS, both based on the transformer architecture but with different attention mechanisms that lead to the ability to process documents of different lengths.

The results are evaluated through two metrics: ROUGE, a metric that favours the syntactic similarities by the generated text and the one taken as ground truth, and BERTScore, which privileges the semantic similarities.

Although the ROUGE metric generally leads to higher results for texts generated with extractive summarization techniques, the obtained results are significantly high for an abstractive summarization task. In particular, they are higher than the results obtained in previous works.

The results show that the input section deeply affects the first claim generation performance. The best input text turns out to be the combination of summary and abstract, whereas the least informative section is the description.
In all the cases BigBird-PEGASUS, the model that processes longer documents, leads to higher performances, at the expense of the training time that is almost three times that of PEGASUS.

Table of Contents

List of Tables

List of Figures

Acronyms

FY

Fiscal Year

USPTO

United States Patent and Trademark Office

GSG

Gap Sentence Generation

MLM

Masked Language Modeling

PEGASUS

Pre-training with Extracted Gap-sentences for Abstractive SUmmarization Sequence-to-sequence

NLP

Natural Language Processing

HUPD

Harvard USPTO Dataset

RNN

Recurrent Neural Network

IP

Intellectual Property

ROUGE

Recall-Oriented Understudy for Gisting Evaluation

BERT

Bidirectional Encoder Representations from Transformers

GPT

Generative Pre-trained Transformers

LED

Longformer Encoder-Decoder

API

Application Programming Interface

GPU

Graphics Processing Unit

Chapter 1

Introduction

The Latin word *patere*, which means "to be open" (i.e., to allow public inspection), is where the word patent originated. It is a shortened version of *letters patent*, which is a type of legal document published by a president, monarch, or other head of state in written form. It was typically used to bestow an office, right, monopoly, or title on an individual or organization.
A patent is a kind of intellectual property that grants its owner the authority to exclude others to make, sell, or use an invention for a fixed amount of time in exchange for sufficient disclosure of the invention. With these premises, it can be understood how the patent system has developed intending to foster innovation and economic development. Indeed, by granting exclusive rights for a limited period, an inventor can recoup R&D expenses and investments. Additionally, the system makes knowledge and information available to the general public [1].

Each patent is made of five main parts [2]:

- **Abstract**: the abstract aims to allow the public and the USPTO to quickly determine the technical disclosure of the invention. It also underlines what is new in the art field to which the invention belongs.
- **Background**: a description of the area of endeavour to which the invention relates should be included in this section. A paraphrase of the relevant U.S. patent classification definitions or the topic of the claimed invention may also be included in this part;
- **Summary**: This section should give a concise summary of the claimed invention's main points or concepts. The advantages of the invention and how it resolves problems that already existed can be included in the summary;
- **Description**: in this section, the invention must be fully stated, along with how to make and use the invention, in full, clear, concise, and exact words.

This section should distinguish the new invention from other and old inventions, and should also describe the method, the machine, the manufacturing, the composition of matter, or the improvement that has been created;

- **Claims**: they must specifically identify and assert the elements that the inventor or inventors believe to be the innovation. Each of them is composed of a single sentence. The claims specify the boundaries of the patent's protection. The extent of the claims determines, in large part, whether a patent will be granted or not.

If the patent is accurately written, the first four aforementioned parts are under the responsibility of the inventors, whereas the claims formulation task is up to the Intellectual Property (IP) attorney.
The task the IP attorney has to perform is to deeply understand the patent, locate the most important information and identify the boundaries of the patent's protection. This task in legal terms is called *claim construction* [3].

The most important claim is the first one. It is composed of a single sentence, has to set out the distinctive features of the invention and underline the differences from the inventions already present in the same or similar field. It also should only include the essential features of the invention and define the level of restriction desired [4].

The claims are frequently phrased in legal jargon, which makes it challenging for non-experts to comprehend them. Also, the rate of patent applications is rising rapidly today. The USPTO received more than 665,000 patent applications in fiscal year 2019, nearly twice as many as it did in fiscal year 2002. Even with such a significant increase, the USPTO continued to adapt and recently met the FY2018–2022 Agency Priority goal of a first-time pendency of under 15 months and a total pendency of under 24 months [5].

This explosion of submissions, and the complexity of the analysis task, led to the main problem of finding ways to fasten the analyses of patents in order to keep up with the innovations. A possible way to process all this data is to exploit the Natural Language Processing (NLP).

Computer science's NLP discipline focuses on how computers and human languages interact. NLP is used to create tools and algorithms that let computers decipher, interpret, and produce human language. In recent years, as the volume of unstructured text data has exponentially increased, its significance has grown [6]. Analyzing this unstructure data can be time-consuming and challenging, as it requires a deep understanding of language and context. NLP techniques can help

address these challenges by enabling computers to extract meaning and insights from unstructured text data. For example, NLP can be used to generate summaries and abstracts of large volumes of text, making it easier for humans to quickly understand the most important information contained in the documents.

The IP attorney duty can be thought of as a summarization task. Indeed, the task requires one to first comprehend and interpret the input document. After that, the attorney has to give higher priority to the most informative parts compared to the context information, and then, paraphrase the important content into a fluent and coherent re-elaborated version.
More specifically, human summarizers have four main characteristics:

1. they are able to comprehend and interpret the input document;
2. they give a higher priority to the most informative parts compared to the context information;
3. they paraphrase the important content into a fluent and coherent re-elaborated version;
4. they are able to generate multiple summaries given the same input document.

The NLP scope provides several techniques to summarize input texts. The NLP summarization scope can be divided into two big families of models: extractive summarization models and abstractive summarization ones. Extractive summarization techniques extract and concatenate the most informative sentences into a single summary, these techniques are well suited to discovering the most important information, but they generate summaries that may be weak in terms of coherency and fluency. Abstractive summarization instead shows the ability to re-elaborate the input text into a shortened version, containing the most important information, keeping the keywords, but without coping fragments from the input document. Therefore, they are more suited to address the fluency and coherency problem, in addition to the ability of generating multiple summaries starting from the same input text. When the summarization task is taken to its extreme, as shortening the input text into a single-sentence summary, it is possible to talk of *extreme summarization*.

Patents, as compared to other types of texts, for instance in the news domain, have a richer discourse structure, are full of legal terms, and the informative content tends to be more evenly distributed, whereas, in a typical news article, the first part is the most informative one [7]. This different structure has set new challenges concerning the summarization task. Patent summarization works aim to obtain NLP models that are capable to generate patent sections, using the correct technical

terms, identifying the distributed important content and generating fluent new text. The underlying idea is to speed up the inventors writing or the IP attorney analysis.
Most previous patent summarization works are focused on using the claims section or the description as a source of information to generate a patent section such as the abstract or the summary. More in detail, the BIGPATENT dataset (Section 3.2.1), one of the most well-established patent datasets, allows only to use of the patent description as a source of information, trying to generate the summary section. A brand new dataset, the Harvard USPTO Dataset (HUPD) (Section 3.2.3), presented in 2022, contains 34 fields for each patent, allowing deeper and wider patent analyses. The HUPD authors, for what concerns the summarization task, focused on two experiments. One uses the description section as input, and the other the claims. In both cases, the aim was to generate the abstract section. A third important work, related to the patent summarization scope, is presented by the authors of the CMUmine dataset (Section 3.2.2). This dataset contains all five aforementioned sections, and the authors performed the summarization task using the summary section as a source of information, trying to generate the first claim. This is the only work that speeds up the IP attorney's work.

In this thesis, the goal is to generate the first claim of a patent document using abstractive summarization techniques that can understand the context and meaning of the input text, and generate fluent and coherent first claims.
The study focuses on two main research questions:

- which patent sections are the most effective for generating the first claim;
- how does the length of the input text impact the performance of the summarization models.

This research could have significant implications for the legal and innovation communities by improving the accuracy and efficiency of automated patent claim generation.
Seven different input texts are analyzed, including single sections like abstract, background, description, and summary, as well as combinations of two sections. The boundaries of each section are highlighted using special tokens to help the models recognize the semantic content. To investigate the impact of context width, two models are compared: PEGASUS and BigBird-PEGASUS, both based on the transformer architecture but with different attention mechanisms. The experiments are performed on the CMUmine dataset, which contains around 300,000 patents. See Section 5 for more details.

The results are evaluated through two metrics, ROUGE, a metric that favours the syntactic similarities by the generated text and the one taken as ground truth,

and BERTScore, which privileges the semantic similarities (Section 6.1). Although the ROUGE metric generally leads to higher results for texts generated with extractive summarization techniques, the obtained results (Section 6.3) are significantly high for an abstractive summarization task. In particular, they are higher than the results obtained in previous works. The results show that the input section deeply affects the first claim generation performance. In particular, the best input text turns out to be the combination of summary and abstract, whereas the best single-section tested is the summary, which leads to performance slightly lower than those obtained using the aforementioned text combination. The analysis led to discover that the least informative section is the description one, followed by the background. A possible explanation could be that the description section is significantly longer and using any of the two tested models the text is truncated. In all the cases, that is with each input text analyzed, BigBird-PEGASUS, the model that processes longer documents, leads to higher performances, unfortunately at the expense of the training time that is almost three times that of PEGASUS. In particular, while PEGASUS needs an average of 20 hours to be trained, BigBird-PEGASUS reaches three days of training. Due to the time limits imposed by the computational resources longer experiments have not been done.

This thesis presents the bases for several further analyses. The code presented is modular and allows for the testing of further input texts and section combinations. It is also easily adaptable to different datasets and models. A point of sure interest is to understand whether the generation of claims other than the first one, or the generation of more than one claim at a time, could be tasks that can be addressed with the method presented in this thesis, or the generation of a longer text needs a different strategy. Another future plan is to test other models that allow to process of even longer input text. Indeed, models with wider context could be able to process the entire description section or the combinations of more than two sections. It is also interesting to understand if longer training could lead to significantly higher performances. A further analysis that can be performed with different computational resources is to apply the presented approach to the HUPD dataset, understanding if a dataset with a cumbersome amount of data as HUPD, with about 4.5 million patents, could lead to effective performance improvement compared to a smaller one, as CMUmine.

Outline of the thesis

In Section 2 a detailed overview of the embeddings is reported, starting with the definition of embedding and ending with the differences between static and contextualized embeddings. This section also contains a detailed description of the two summarization models used for this thesis.

Section 3 describes the general characteristics of some common datasets for text summarization, then focuses on the datasets specific to the patent domain.

In section 4 the most famous abstractive summarization models are analyzed and divided into general-purpose and task-specific ones. Then, the analysis is focused on the already applied approaches for patent summarization.

Section 5 deeply describes all the steps performed in this thesis, from the preparation of the datasets to the input creation with the new special tokens, ending with training and test parameters.

Section 6, the Results section,vextensively analyzes the obtained results, making deep comparisons between the obtained performances.

The last part, Section 7, reports the conclusions of this thesis and some future perspectives.

Chapter 2

Background

This section is dedicated to explaining the previous knowledge necessary to understand this thesis work. In particular, after an overview on the embeddings, the first part is dedicated to the language modelling problem making a distinction between context-independent and contextualized embeddings. Then, in Section 2.2 the used models, PEGASUS and BigBird, are described in detail.

Embeddings are a type of latent representation, which refers to a lower-dimensional representation of a high-dimensional data space that captures meaningful information about the data. Embeddings are used extensively in machine learning and artificial intelligence, particularly in computer vision, audio, and natural language processing.

In computer vision, embeddings are often used to represent images. Instead of using the raw pixel values of an image, an embedding model is trained to map the image to a lower-dimensional vector space that captures meaningful information about the image, such as its content, color, texture, and shape. These embeddings can then be used for various tasks, such as image classification, object detection, and image retrieval.

In audio processing, embeddings are used to represent audio signals, such as speech or music. Similar to computer vision, an embedding model is trained to map the audio signal to a lower-dimensional vector space that captures meaningful information about the audio, such as its pitch, tone, and timbre. These embeddings can then be used for various tasks, such as speech recognition, music genre classification, and speaker identification.

In natural language processing, embeddings are used to represent words or phrases. Instead of using the raw text of a document, an embedding model is

trained to map each word or phrase to a lower-dimensional vector space that captures its semantic and syntactic properties. Typically, similar words in meaning are expected to be closer in the vector spece [8].

Overall, embeddings have proven to be a powerful technique for representing complex data in a lower-dimensional space that captures meaningful information. They have enabled breakthroughs in computer vision, audio processing, and natural language processing, and are a critical component of many state-of-the-art machine learning models.

This thesis, working on texts, is focused only on the word embeddings.

2.1 Word embeddings

Word embeddings are numerical representations of words that capture their semantic and syntactic information. There are two main types of word embeddings: static and contextualized.

Static word embeddings, also known as pre-trained word embeddings, are learned from a large corpus of text and are fixed in their representation of each word. These embeddings are trained using unsupervised methods and can be used across different tasks. Examples of popular static word embeddings include Word2vec, fastText, and GloVe.

Contextualized word embeddings, on the other hand, take into account the context in which a word appears and produce a different representation for each instance of a word. These embeddings are trained using supervised methods, often on large language models, and can capture more complex relationships between words in a sentence. Examples of popular contextualized word embeddings include ELMo, BERT, and GPT.

The key difference between static and contextualized word embeddings is that static embeddings are trained on a fixed representation of a word and are the same for every context, while contextualized embeddings are able to capture the meaning of a word in the context of the sentence or document in which it appears.

In practice, which type of word embedding to use depends on the specific task and the amount of data available. For tasks that require a high degree of understanding of the context, such as sentiment analysis or text summarization, contextualized embeddings are typically more effective. For tasks where the focus is on word-level relationships and analogies, such as word similarity or analogy completion, static embeddings may be sufficient.

2.1.1 Static word embeddings

Word2vec

Word2vec was introduced in 2013 by Tomas Mikolov and his colleagues at Google [9]. The algorithm was designed to address the limitations of traditional NLP techniques, for instance, one-hot encoding, which relied on hand-crafted features and statistical models. Word2vec is a neural network-based algorithm that uses a simple yet powerful architecture to learn word embedding.

There are two main architectures for implementing Word2vec: the Continuous Bag-of-Words (CBOW) architecture and the Skip-Gram architecture. In the CBOW architecture, the model predicts a target word based on its context, which is a set of surrounding words. In contrast, the Skip-Gram architecture predicts the context words given a target word. Both architectures use a simple neural network with one hidden projection layer to learn word embeddings.

The basic idea behind Word2vec is to train a neural network to predict the probability of a word given its context, or vice versa. The Skip-Gram network takes in a one-hot encoded vector representation of the input word and maps it to a lower-dimensional embedding vector through the hidden layer. The hidden vector is then passed through an output layer to generate a probability distribution over the vocabulary. This process is repeated for each word in the training corpus. In the CBOW network instead, the one-hot representations of the context words passed to the projection layer shared for all words, are mapped into a hidden vector. The obtained vector is forwarded to the output layer. This last layer returns the probability distribution over the vocabulary. The two presented architectures are depicted in Figure 2.1

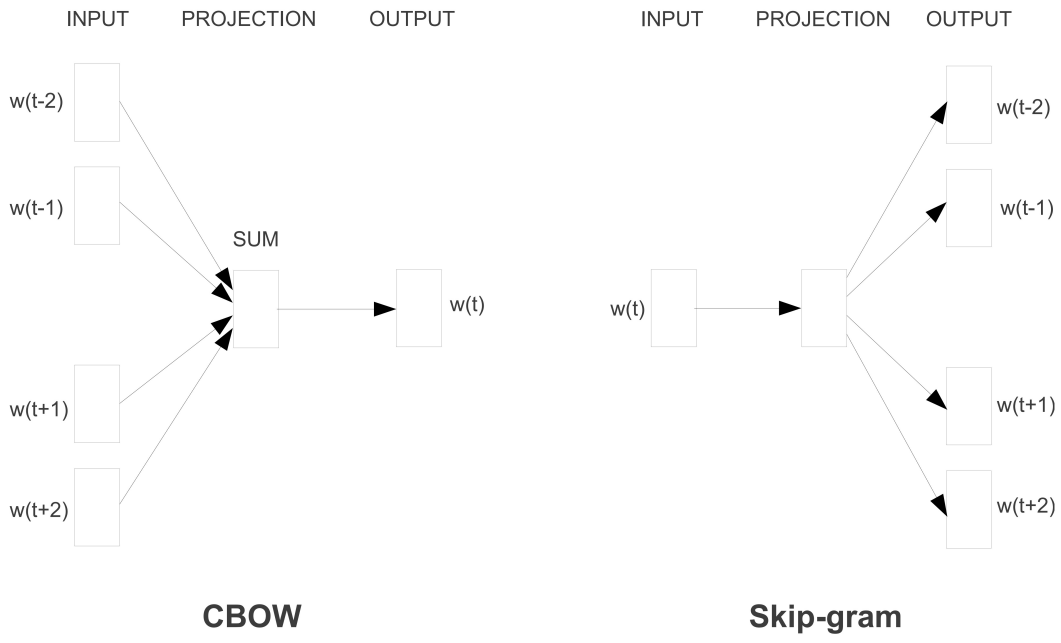


Figure 2.1: The Continuous Bag-of-Words (CBOW) architecture and the Skip-Gram architecture presented in [9].

The objective of the Word2vec algorithm is to maximize the likelihood of the

training data given the model parameters. This is achieved by minimizing the negative log-likelihood of the training data, which is equivalent to maximizing the average log probability of the context words given the target word or vice versa:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{2.1}$$

In Equation (2.1) $w_1, w_2, ..., w_T$ is a sequence of words of cardinality T, c is the number of words to consider that are placed before and after the target word.

The word embeddings generated by the Word2vec algorithm are analyzed by the authors to understand the semantic and syntactic relationships between words. The analysis showed that words with similar meanings are clustered together in the vector space, while words with opposite meanings are separated. The word embeddings also capture the syntactic relationships between words, such as verb tense and noun-verb relationships.

One of the main advantages of Word2vec is its ability to capture the semantic and syntactic relationships between words. However, the algorithm also has some limitations, such as the inability to handle out-of-vocabulary words, which can be handled using FastText, the ignoring of global word co-occurrence, a problem that can be solved using Glove and the inability to encode contextual representations, which can be encoded using contextualized embeddings.

FastText

FastText is a simple and efficient neural network-based model for language modelling, developed by Facebook AI Research in 2016 [10]. The FastText model is based on the word embedding technique that represents each word in a continuous vector space. In addition to word embeddings, FastText also uses character n-grams to capture subword information. The model takes as input a sequence of words, and for each word, it computes a vector representation by summing the embeddings of the individual word and character n-grams. The final word vector representation is associated with a score. Suppose to have a dictionary of n-grams of size G. Given a word w, denote as $\mathcal{G}_w \subset 1, ...G$ the set of n-grams contained in w. Associate to each n-gram g its vector representation $\mathbf{z}_g$. The used scoring function is:

$$s(w, c) = \sum_{g\in\mathcal{G}_w} \mathbf{z}_g^T \mathbf{v_c} \tag{2.2}$$

FastText can capture subword information using character n-grams, which makes it robust to misspellings and out-of-vocabulary words. It is also capable of handling

multilingual data, making it a useful tool for cross-lingual applications. However, the model has limitations in handling long-term dependencies, and it may not perform well on tasks that require modelling the context over a long sequence of words.

GloVe

GloVe, Global Vectors for Word Representation, is an unsupervised learning algorithm that was introduced in 2014 by Pennington et al [11]. The key idea behind GloVe is to leverage the co-occurrence statistics of words in a corpus to learn a low-dimensional vector representation for each word. Unlike other neural language models, which use a sliding window approach to capture local context, GloVe is designed to model global context by taking into account the entire corpus. Formally, given a corpus of size V, the co-occurrence count between two words i and j is defined as X_{ij}, which is the number of times i and j co-occur in a context window. The GloVe algorithm aims to learn a low-dimensional vector representation w_i for each word i in the corpus, such that the dot product of the vectors for two words i and j is proportional to the log of their co-occurrence probability:

$$w_i \cdot w_j \propto \log(X_{ij}) \tag{2.3}$$

To achieve this, GloVe introduces a weighting function $f(X_{ij})$ that captures the relative importance of the co-occurrence count X_{ij}. The weighting function has to respect three properties:

- $f(0) = 0$;
- $f(x)$ is non decreasing;
- $f(x)$ is relatively small for large values of x.

A common function is:

$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{x_{\max}}\right)^{\alpha} & \text{if } X_{ij} \leq x_{\max} \\ 1 & \text{otherwise} \end{cases} \tag{2.4}$$

where x_max is a threshold that caps the maximum co-occurrence count, and α is a hyperparameter that controls the strength of the weighting function. The use of a weighting function helps to mitigate the effect of noise in the co-occurrence counts. Once the weighting function is defined, the optimization objective of GloVe is to minimize the following cost function:

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i \cdot w_j + b_i + b_j - \log(X_{ij}))^2 \tag{2.5}$$

where b_i and b_j are bias terms for words i and j, respectively.
One of the advantages of GloVe is its computational efficiency. Because it uses global context to learn word embeddings, it is able to leverage large corpora without the need for computationally expensive neural language models. Additionally, the GloVe algorithm is relatively simple to implement, making it accessible to researchers and practitioners who may not have extensive experience with deep learning. Another advantage of GloVe is its interpretability. Because the vectors learned by the algorithm are based on co-occurrence statistics, they have a clear semantic interpretation. Additionally, GloVe embeddings have been shown to outperform other state-of-the-art methods, such as Word2vec, on many NLP tasks such as word analogy, word similarity, and named entity recognition.

2.1.2 Contextualized word embeddings

Traditional word embeddings, such as word2vec or GloVe, do not consider the context of a word, and hence, they cannot capture the multiple meanings of a word in different contexts. In recent years, several pre-trained models have been developed that generate contextualized word embeddings, such as ELMO, BERT, and GPT.

ELMO

ELMO, which stands for Embeddings from Language Models, is a state-of-the-art deep learning model for generating contextualized word embeddings. It was developed by researchers at the Allen Institute for Artificial Intelligence in 2018 [12] and has become increasingly popular in the Natural Language Processing (NLP) community in recent years.

ELMO generates contextualized word embeddings by training on a large corpus of text. It is based on a multi-layer bi-directional language model, which means that it processes the input text both forwards and backwards, allowing it to capture the contextual meaning of words. ELMO consists of three main components: a character-level convolutional neural network, a bi-directional LSTM, and a task-specific feedforward neural network. Bi-directional LSTMs have the sense of both the next and the previous words. They are made of two language models: a forward language model and a backward one. A forward language model computes the probability of the sequence a sequence of tokens by modelling the previous ones:

$$p(t_1, t_2, ..., t_n) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1})$$

A backward language model instead predicts the sequence probability given the

future context:

$$p(t_1, t_2, ..., t_n) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, ..., t_N)$$

Each word embedding is built following three steps: first, concatenate the hidden representations obtained by the forward and the backward language models, then multiply each obtained vector by a task-dependant weight, and finally sum the weighted vectors.

One of the benefits of pre-trained models like ELMO is that they can be used for transfer learning. This means that the model can be fine-tuned on a specific natural language processing task with a relatively small amount of task-specific data. This allows researchers and practitioners to leverage the knowledge and expertise that has been built into the pre-trained model and apply it to their own specific tasks. ELMO has been shown to perform well on various natural language processing tasks. It has been shown to outperform traditional word embeddings on many tasks and to be competitive with other state-of-the-art models like BERT and GPT.

While ELMO has many benefits, it also has some limitations. For example, it requires a large amount of training data, which can be a challenge for some applications. Additionally, it can be computationally expensive to train and use, which can be a limitation for some applications that require real-time processing.

Transformers

Transformers are a class of deep learning models that have revolutionized the field of NLP in recent years. They were first introduced in 2017 in a paper titled "Attention Is All You Need" by Vaswani et al. [13], and have since become the state-of-the-art models for many NLP tasks, such as machine translation, text classification, and text generation. Traditional neural network models for NLP, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), have limitations in their ability to model long-range dependencies in text. Transformers, on the other hand, are able to model long-range dependencies more effectively through the use of attention mechanisms.

The core idea of a transformer model is to use self-attention mechanisms to compute a weighted sum of all the words in the input text, allowing the model to attend to different parts of the input text when making predictions. This makes the model more effective at capturing the relationships between words and the contextual meaning of the text. The attention mechanism starts from three vectors, Key, Value and Query, and outputs the weighted sum of the Values, where the weights are computed according to a compatibility function that measures the compatibility

of the Query with the corresponding Key. One of the most widespread attention operations is given by

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.6}$$

Transformers have also brought about significant improvements in the efficiency of training and inference for NLP models. This is achieved through the use of parallel computation and the ability to process batches of input text simultaneously. This parallel computation requires the introduction of positional embeddings without which the information about the relative and absolute positions of words would be lost. The used positional embeddings involve sine and cosine functions at different frequencies:

$$\begin{aligned} PE_{pos,2i} &= sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \\ PE_{pos,2i+1} &= cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \end{aligned} \tag{2.7}$$

where d_{model} is the embedding dimension.
The Transformers [13] architecture is composed of an encoder-decoder structure which makes use of the attention mechanism. The encoder is made of N identical layers, each one divided into two sublayers:

1. Multi-head self-attention: multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions;

$$MultiHead(Q, K, V) = concat(head_1, head_2, ..., head_N)W^O$$

$$head_i = Attention(QW_i^Q, KW_i{}^K, VW_i^V)$$

   where the *Attention* is given by Equation 2.6;

2. Position-wise feed-forward neural network: this layer consists of two linear layers with a ReLU activation in between.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

Residual connections followed by a normalization layer are added around each sublayer.

For what concerns the decoder stack, has N identical layers, each composed of three sublayers:

1. Masked multi-head self-attention: a mask ensures that the predictions for position i can depend only on the known outputs at positions less than i;

2. Multi-head encoder-decoder attention: performs attention on the output of the encoder stack;

3. Position-wise feed-forward neural network.

Also in the decoder stack, there are residual connections followed by a normalization layer around each sublayer. The transformer's architecture is represented in Figure 2.2

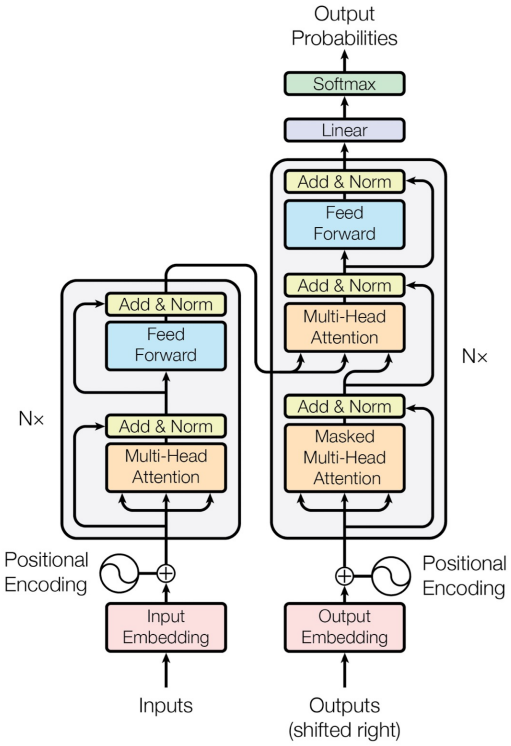


Figure 2.2: The encoder-decoder Transformer's architecture explained in [13].

BERT

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based neural network model that was introduced by Google researchers in 2018 [14]. BERT is a powerful tool for natural language processing that has gained popularity due to its ability to generate contextualized word embeddings.
BERT is a multi-layer bidirectional transformer encoder. The bidirectional aspect of BERT means that the model can process the input text in both directions, which allows it to capture the contextual meaning of words in the text.

BERT has two pre-training phases. In the first phase, called the masked language model (MLM), the model is trained to predict a randomly masked word in the input text. More specifically, this model uniformly selects 15% of the input tokens for possible replacement. Of the selected tokens, 80% of them are replaced with the [MASK] token, 10% are left unchanged and 10% are replaced by a randomly selected token in the vocabulary.
In the second phase, called the next sentence prediction (NSP), the model is trained to predict whether two sentences in the input text are consecutive or not. This pre-training phase allows the model to learn semantic dependences among sentences.
After pre-training, BERT can be fine-tuned on specific natural language processing tasks with relatively small amounts of task-specific data.
To use BERT, we first tokenize the input text into a sequence of word pieces. These word pieces are then converted into numerical representations using an embedding layer. The resulting numerical representations are fed into the BERT model, which generates a sequence of contextualized word embeddings.
One of the limitations of BERT is its computational requirements. BERT is a large model that requires significant computational resources to train and use effectively. This can be a challenge for smaller organizations or individuals who do not have access to large-scale computing resources.

GPT

Generative Pre-trained Transformer (GPT) is a transformer-based deep learning model for generating contextualized word embeddings. It was introduced by OpenAI in 2018 [15] and has become a popular tool for natural language processing tasks.
GPT is a multi-layer transformer decoder that is pre-trained on a large corpus of text using a language modelling objective. Unlike BERT, which is a bidirectional model, GPT is a left-to-right model that generates text one word at a time.
The pre-training objective for GPT is to predict the next word in a sequence of text.
To use GPT, first, tokenize the input text into a sequence of word pieces. These word pieces are then converted into numerical representations using an embedding layer. The resulting numerical representations are fed into the GPT model, which generates a sequence of contextualized word embeddings.
One of the strengths of GPT is its ability to generate high-quality text that is both grammatically correct and semantically meaningful. This makes it a powerful tool for tasks such as text generation and language translation.

2.2 Summarization models used

PEGASUS and BigBird-PEGASUS are both transformer-based models that have been developed by Google for natural language processing tasks. PEGASUS is primarily designed for text summarization, where it excels in generating accurate and readable summaries of text. BigBird is a sparse attention mechanism designed to handle long sequences of text more efficiently, making it well-suited for a wide range of NLP tasks.

2.2.1 PEGASUS

PEGASUS is a state-of-the-art pre-training-based text-to-text neural network architecture that has achieved impressive results in a variety of NLP tasks. It was introduced in a research paper by researchers at Google in 2019 [16] and has since become one of the most popular NLP models for natural language generation, summarization, and other related tasks.
At its core, PEGASUS is a variant of the Transformer architecture, which is a type of neural network architecture that uses attention mechanisms to model the relationships between input and output sequences. However, unlike other Transformer-based models that are trained to predict the next word in a sequence, PEGASUS is trained using a pre-training-based approach that involves generating a summary of a document called Gap Sentence Generation.
One of the key advantages of PEGASUS is its ability to generate summaries that are not only accurate but also fluent and readable.

There are two versions of PEGASUS available: $\text{PEGASUS}_{\text{BASE}}$ and $\text{PEGASUS}_{\text{LARGE}}$. The larger model has more attention heads, from 12 in its base form to 16, more layers in each encoder and decoder, from 12 to 16, and a bigger hidden size. This thesis makes use of the $\text{PEGASUS}_{\text{LARGE}}$ model. PEGASUS can handle sequences of up to 1,024 tokens.

GSG With PEGASUS [16] a new self-supervised objective is introduced: the Gap Sentence Generation. The idea is that putatively important sentences are masked in the original text and reconstructed in output by the model, using only the remaining sentences. This objective is particularly appropriate for abstractive summarization as it closely resembles the downstream task. The sentences considered important are chosen on the basis of their ROUGE1-F1 score, computed against the remaining text. This selection technique leads to significantly higher performances with respect to lead or random sentence selection. The PEGASUS base model uses also the MLM objective, thus it also masks a random set of tokens. However, this technique is shown to lead to no improvements, therefore in the PEGASUS large

model, only the GSG objective is preserved.

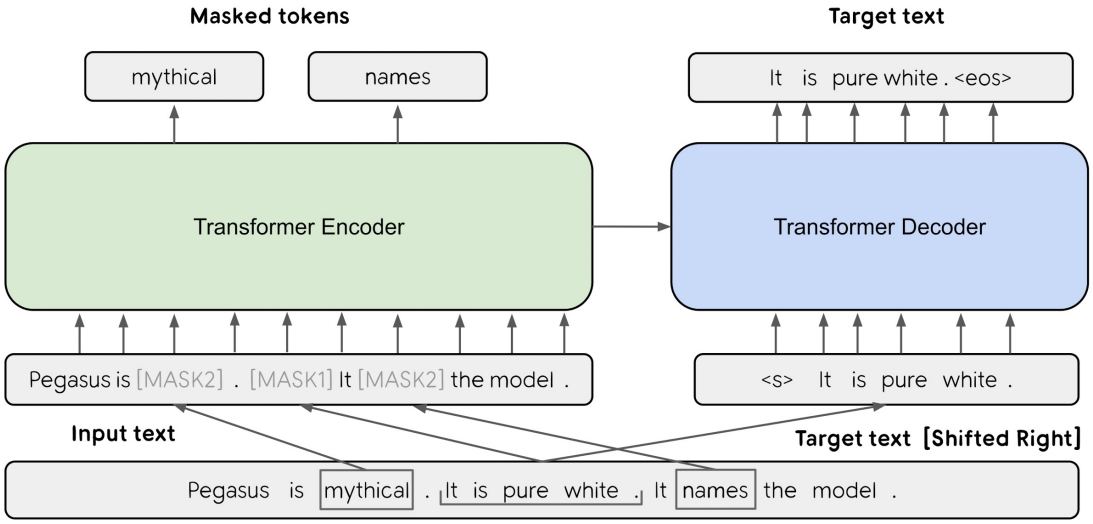


Figure 2.3: PEGASUS training objective explained in [16]. Both GSG and MLM are shown, although only the first one is used by the model exploited in this thesis.

2.2.2 BigBird

The full attention mechanism typical of transformers leads to their main limitation: the quadratic dependency, mainly in terms of memory, with the input length. The self-attention, indeed, is a double-edged sword, it allows us to overcome the RNN restrictions allowing all the input tokens to attend to all the other tokens in the sequence, but this innovation leads to heavy memory problems.

BigBird is a sparse attention which extends Transformer based models extending their ability to efficiently process long sequences of text. Introduced by Google researchers in 2020 [17], BigBird is designed to handle long-range dependencies in text data. Indeed, BigBird uses a novel sparse attention mechanism that allows it to process long sequences of text in a more efficient way, reducing the dependency from quadratic to linear.

BigBird main contributions to the attention mechanism are three:

1. a set of g global tokens that attend on all the parts of the sequence;
2. all tokens attend to a set of w local neighbouring tokens;

3. all the tokens attend to a set of r random tokens.

Given an input sequence $X = (x_1, x_2, ..., x_n) \in \mathbb{R}^{n \times d}$ the sparse attention mechanism can be described by a directed graph D with a set of nodes $[n] = \{1, ..., n\}$ and a set of edges that represent the inner products that the attention will consider.
The new attention mechanism can be defined as:

$$Attention_D(X)_i = x_i + \sum_{h=1}^{H} \sigma\left(Q_h(x_i) \cdot K_h(X_{N(i)})^T\right) \cdot V_h(X_{N(i)}) \tag{2.8}$$

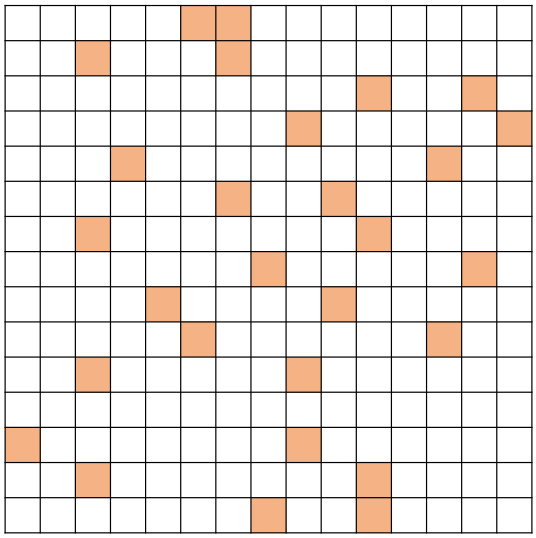
where $N(i)$ is the set of out-neighbours of node i, H is the number of attention heads, σ is the softmax scoring function and $X_{N(i)}$ is the matrix obtained stacking $\{x_j : j \in N(i)\}$ and not the entire input.
In the case in which the graph D is a complete graph, the full quadratic attention mechanism is recovered.

The arcs considered to obtain the sparse attention mechanism are now explained by means of the adjacency matrix A for ease of comprehension.
The three aforementioned viewpoints are examined:

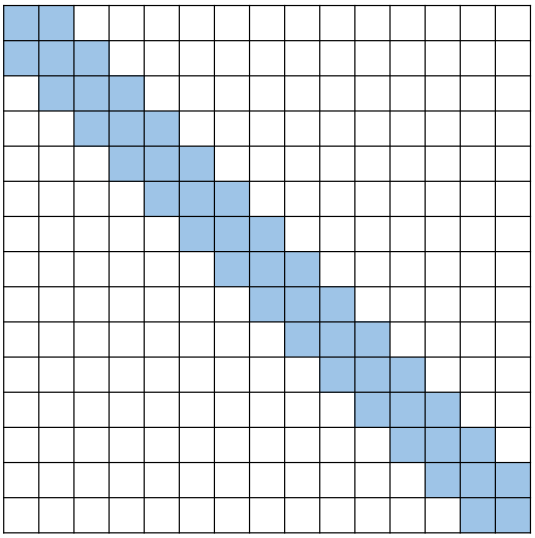
- *Random tokens*: each query attends over r random keys. In that way $A(i, \cdot) = 1$ for each key chosen in a random fashion (Figure 2.4a);
- *Sliding window*: in the majority of contexts in NLP a great deal of information can be derived by the neighbouring tokens. To handle this information the Watts and Strogatz model [18] is used: each node is connected with w neighbours, $w/2$ on each side. Consequently the adjacency matrix will be: $A(i, i - w/2 : i + w/2) = 1$ (Figure 2.4b). Thus, the self-attention of width w at location i attends keys from $i - w/2$ to $i + w/2$;
- *Global tokens*: a set of existing tokens G are made general, they attend to every token in the sequence and to whom each token attends. To achieve this idea the adjacency matrix will be built as $A(i, :) = 1$ and $A(:, i) = 1 \ \forall i \in G$ (Figure 2.4c).

BigBird merges all these viewpoints obtaining the attention shown in Figure 2.4d. This approach allows BigBird to handle sequences of up to 4,092 tokens, which is much longer than what most other models can handle.

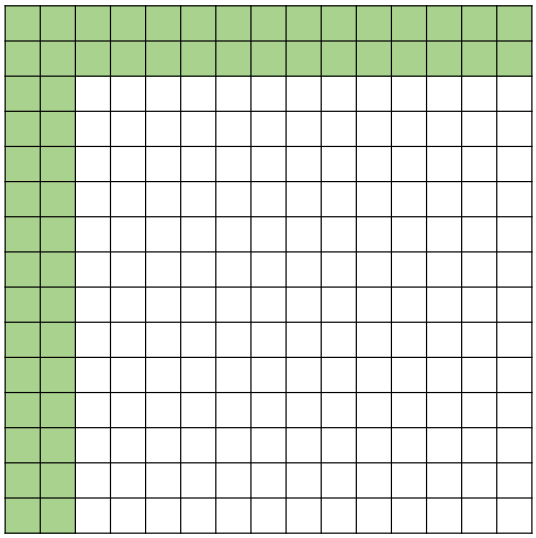
The authors have made available two models, one that leverages RoBERTa [19], BigBird-RoBERTa, the smaller one, and the other that leverages PEGASUS [16], BigBird-PEGASUS, the bigger one. To be consistent with the experiments, in this thesis the focus is set on the second bigger model, BigBird-PEGASUS.



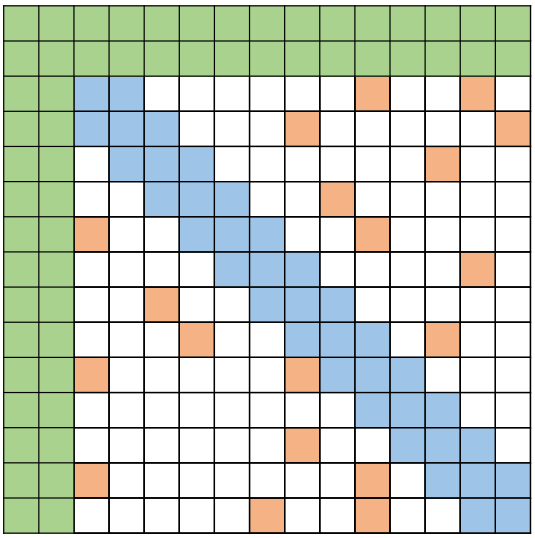
(a) Random tokens



(b) Sliding window



(c) Global tokens



(d) BigBird sparse attention.

Figure 2.4: Building blocks of sparse attention mechanism presented in [17]. Figure (a) shows a random selection of keys with $r = 2$. Figure (b) represents the sliding window with $w = 3$. Figure (c) depicts the selected $g = 2$ global tokens. Figure (d) reports the final BigBird sparse attention mechanism.

Chapter 3

Datasets

General text summarization datasets and patent summarization datasets differ in terms of their domain and cont.

Domain: General text summarization datasets are sourced from a wide range of domains, such as news articles, scientific papers, and social media posts. These datasets aim to provide a broad view of the summarization task, which is applicable to a wide range of domains. Patent summarization datasets, on the other hand, are sourced specifically from patents. They focus on summarizing the content of patent documents, which can be highly technical and specialized.

Content: General text summarization datasets contain text that is typically written in a more natural language style and the vocabulary is often more familiar to the average reader. Patent summarization datasets, on the other hand, contain highly technical language. The text often includes specialized terminology, legal terms, acronyms, and abbreviations that may not be familiar to the average reader.

3.1 Benchmark datasets for text summarization

There are several benchmark datasets available for text summarization that are widely used for evaluating the performance of automatic summarization models.

CNN/Daily Mail The CNN/Daily Mail dataset is a widely used benchmark dataset for text summarization, specifically for evaluating abstractive summarization models. The dataset consists of news articles from CNN and Daily Mail paired with bullet-point summaries and is designed to test the ability of models to generate human-like summaries that capture the most important information in the text. Bullet-point summaries are treated as multi-sentence summaries, where each bullet is considered a sentence. One of the challenges of the CNN/Daily Mail dataset is that it contains long and complex articles, which can be difficult to summarize accurately [20].

Gigaword The Gigaword dataset is a large-scale text corpus that is widely used in natural language processing research, particularly for text summarization. The dataset contains news articles from the New York Times and Associated Press, as well as their corresponding one-sentence article summaries. One of the advantages of the Gigaword dataset is that it provides a large and diverse set of documents, which allows researchers to test the robustness and generalization capabilities of their models. The articles in this dataset are similar in length to those in the CNN/Daily Mail dataset, but the summaries are shorter and more focused. The dataset is often used to evaluate extractive summarization models, and more recently, abstractive summarization models [21].

PubMed The PubMed dataset is a widely used corpus in natural language processing research, particularly in the biomedical domain. The dataset consists of scientific articles from the PubMed Central database, which is a repository of biomedical research articles maintained by the National Institutes of Health (NIH) in the United States. The PubMed dataset is often used to evaluate text summarization models in the biomedical domain, including both extractive and abstractive summarization models. The summaries are typically written by the authors of the articles and provide a brief overview of the study's purpose, methods, results, and conclusions. One of the challenges of the PubMed dataset is that the articles can be highly technical and complex, with specialized terminology and domain-specific jargon. This makes it difficult to develop summarization models that can accurately capture the most important information in the articles. Additionally, the reference abstracts can be highly abstractive, which can make it difficult to evaluate the performance of extractive summarization models that aim to extract important sentences from the article [22].

Newsroom The Newsroom dataset consists of news articles from a wide range of sources, including The New York Times, The Washington Post, and Reuters, and their corresponding human-written summaries. It news articles and their summaries, covering a wide range of topics including politics, business, sports, and entertainment. Unlike other text summarization datasets, such as the Gigaword dataset, which includes mostly extractive summaries, the Newsroom dataset provides more abstractive summaries, making it a suitable dataset for evaluating the performance of abstractive summarization models. The summaries in the dataset are generated by human editors and provide a high-level overview of the article's main points, rather than simply extracting key sentences from the article [23].

XSum This dataset contains news articles from the BBC, along with their corresponding summaries. The dataset is notable for its short summaries, which typically consist of only one or two sentences, and is often used to evaluate abstractive summarization models. The XSum dataset is a challenging benchmark for abstractive summarization models, as the short summaries require models to generate a highly condensed and informative summary that captures the most important information in the article [24].

Overall, the main differences between these datasets are the length and style of the articles, the abstractive nature of the summaries, and the domain in which the dataset is focused. The CNN/Daily Mail, Gigaword, Newsroom and XSum datasets are focused on news articles. The PubMed dataset is focused on biomedical research articles, with author-generated summaries. The Newsroom dataset features a variety of news articles with more abstractive summaries, while the XSum dataset is focused on short news articles with highly abstractive summaries. Researchers should choose a dataset based on the research question and task at hand.

3.2 Patent summarization datasets

Patent summarization is an important task in natural language processing and information retrieval, as patents are a valuable source of technical knowledge and intellectual property. Patent documents can be extremely long and complex, often consisting of highly technical language, making it difficult for humans to sift through the information and identify key points. This is where patent summarization datasets come in, providing a collection of patent documents along with their corresponding summaries to aid in the development and evaluation of automatic summarization models. Indeed, most existing datasets come from the news domain, where the most informative parts are located at the beginning of the text, and where the summarization task leads to copying a lot of extractive fragments.

In this section 3 datasets are presented: BigPatent, CMUmine and HUPD. BigPatent contains only two fields for each patent, however, this dataset is the one used to pre-train the used summarization models. CMUmine is the dataset on which all the tests presented in Section 5 are performed. The HUPD dataset instead, is a cumbersome dataset. Due to its huge size, it could not have been used for computations, but it is of high interest for future works. In light of this, the dataset is prepared to be easily fed to the presented code.

3.2.1 BigPatent

The BigPatent dataset [7] has 1.3 million U.S. patent documents, collected from the Google Patents Public Dataset through BigQuery, which spans a variety of technical domains. More specifically, this dataset contains patents filed after 1971 across nine technological areas, including human necessities, mechanical engineering, chemistry, and more. In order to obtain consistent writing and formatting style, the authors considered only the patents from the USPTO filed in the English language.

With respect to many other summarization datasets, for instance, in the news domain, BigPatent has 3 distinctive characteristics:

- It has summaries with a richer discourse structure;
- The salient content is evenly distributed across all the input text;
- Lesser and shorter extractive fragments appear in summaries.

The abstract of the patent is considered the gold-standard summary of the patent, whereas the detailed description is considered the input for the summarization task. So, this dataset contains only two fields for each patent.

The BigPatent dataset is distributed as a Hugginface Dataset at the following link: `https://huggingface.co/datasets/big_patent` (last accessed: February 2023).

3.2.2 CMUmine

The CMUmine patent dataset [3] contains about 300k patent documents. This dataset was created by crawling the US Patent and Trademark Office's (USPTO) bulk data files, retrieving applications filed between the years 2005 and 2006.

Each patent contains the following information:

- Patent number: the unique identifier associated with each patent;
- Abstract: this section aims to allow the reader to quickly determine what is new in the domain to which the proposed invention pertains;
- Background: it contains a brief description of the area to which the invention is related. It ought to also contain a paraphrase of the relevant patent classification definition and the main topic of the claimed invention;
- Summary: this section contains a brief summary of the invention's main points, it also contains the advantages of the invention and how it addresses already existing problems.;
- Detailed description: this is the longest section, where the invention is fully explained, along with how it can be made by people sufficiently skilled in the art, and also how it can be used. The terms used are precise, clear and concise. This section also explains the novelty of the invention concerning the already existing inventions;
- Claims: this section contains all the claims of the patent, the claims that define the elements the inventor believes to the real innovation, they define the boundary of the protection granted to the patent. In large part, whether a patent will be granted or not depends on the claims;
- First claim: this section contains only the first claim, which is a particular claim that states the distinctive features of the invention, underlining the main differences with the existing art, and defining the restriction level desired.

The authors' aim was to generate the first claim, addressing the problem as an abstractive summarization task, taking as input the summary section. With this perspective, they cleaned the dataset by preserving only the patents with a summary length between [150, 500] and the first claim length between [35, 300]. The dataset can be downloaded at the following link: `https://drive.google.`

`com/drive/u/0/folders/1J4sAcM_21G39VuZT1jv6RqLTEM_UngWS` (last accessed: January 2023).
It is already divided into training, validation and test split with 253,976, 31,736 and 31,696 documents respectively.

Unfortunately not all the patents own a detailed description section, about 1/3 of documents in each split lack the description. More specifically, in the training set 168,456 out of 253,976 documents have the description, in the validation set 20,979 out of 31,736, whereas in the test set 20,960 out of 31,696.
Figure 3.1 shows the situation.

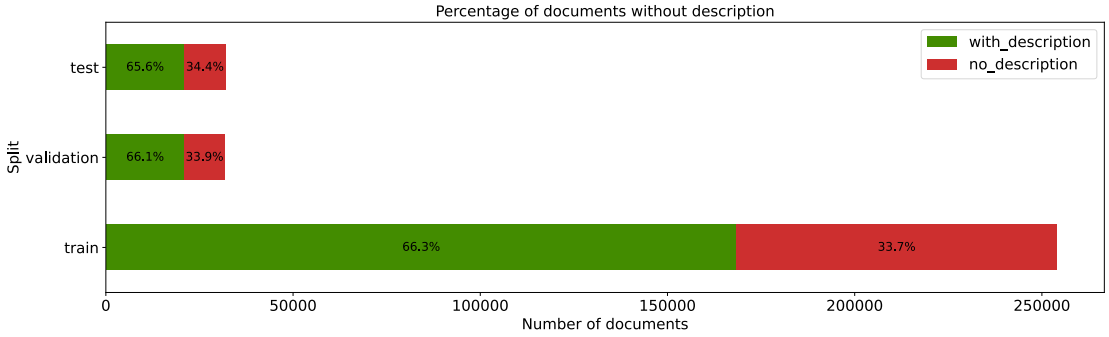


Figure 3.1: Percentage of CMUmine documents without the description field. For each split 2/3 of the documents possess a detailed description.

The dataset is studied in terms of the average number of tokens of each section and the maximum length reached by 99% of the sections. This is done to better understand the problem, and to match the models' constraints about the maximum input lengths.
To count the tokens the text has been split on spaces, dashes and underscores. The obtained results are shown in Table 3.1. Notice that the description results are obtained by excluding the documents that do not own one.

3.2.3 HUPD

The HUPD (Harvard USPTO Patent Dataset) is a dataset of patent documents that are used for research on various natural language processing tasks, such as document summarization, but also text classification, and entity recognition. Indeed, in addition to the patent documents themselves, the HUPD dataset also includes a variety of other useful information. This includes information about the inventors and assignees associated with each patent, as well as information about the USPTO examiners who reviewed each patent application. All the metadata contained are particularly useful for researchers working on patent analysis and

CMUmine	Train		Validation		Test	
Section	Avg.	99th	Avg.	99th	Avg.	99th
Abstract	113.37	238.0	113.23	238.0	113.01	237.0
Background	648.39	2664.25	646.39	2616.3	644.15	2617.45
Description	3102.10	19026.90	3103.12	18967.66	3103.30	19066.94
First claim	112.28	274.0	112.49	274.65	112.33	275.0
Summary	562.87	1440.0	564.17	1446.0	559.23	1440.0

Table 3.1: CMUmine dataset statistics: average number of tokens and 99^{th} percentile of number of tokes for each section. The training set is composed of 253,976 documents, the validation set of 31,736 documents, and the test set of 31,696 documents. Notice that the description results are computed considering only the patents that own a description.

understanding. For instance, the USPTO assigns each patent document a series of classification codes based on the technology areas it pertains to. These codes are based on the International Patent Classification (IPC) system and provide a standardized way to categorize patents according to their technical domain.

The HUPD dataset [25] retrieves patent applications filed to the USPTO between 2004 and 2018, and contains more than 4.5 million patents.
This dataset stands out due to four main reasons:

- It is focused on patent applications, not only accepted ones but also those rejected, allowing a completely new task: the binary classification of patent decisions. In other words, it will enable to predict the acceptance likelihood of a patent;
- It has rich textual and structural information present in patents, more specifically, each patent owns 34 fields;
- It has tripled the size of one of the most famous patent datasets: BigPatent [7];
- Despite its sheer size, being retrieved from USPTO applications, this dataset is still clean, comprehensive and well-structured.

The HUPD dataset is distributed as an Huggingface Dataset at the following link: `https://huggingface.co/datasets/HUPD/hupd` (last accessed: January 2023). The dataset is originally divided into train and test, the test has been created by the dataset's authors considering the patents filed in 2017 and 2018. The validation

set, instead, has been created by randomly sampling from the training set the same number of patents of the test set.
After an extensive dataset study, one has noticed that 8.72% of patents, which means 393,844 out of 4,518,040, lack the first claim. Given that this thesis aims to generate precisely the first claim, these faulty patents are removed, also considering that the amount of patents preserved is remarkable.
The remaining patents are analyzed to better understand their structure in terms of average tokens per section and 99^{th} percentile of the number of tokens per section. The obtained values are reported in Table 3.2.

HUPD	Train		Validation		Test	
Section	Avg.	99th	Avg.	99th	Avg.	99th
Abstract	108.85	226.0	108.71	226.0	109.76	219.0
Background	485.58	2488.0	485.58	2483.0	329.93	2100.0
Description	8135.92	41339.0	8127.36	41131.11	10149.60	49083.08
First claim	134.32	550.0	134.79	559.0	149.87	587.0
Summary	694.49	4578.0	694.70	4550.0	860.52	6184.01

Table 3.2: HUPD dataset statistics: average number of tokens and 99^{th} percentile of number of tokes for each section. The training set is composed of 3,544,996 documents, and the validation and test sets have 289,600 documents each.

Overall, both BigPatent and HUPD are distributed as Hugginface Datasets, therefore they are easily downloadable in a standard format with a single line of code. CMUmine instead is published as a set of zip files loaded on Google Drive, this means that more passes are needed to obtain a dataset that can be easily passed to a summarization model. In addition, BigPatent and CMUmine are specifically devoted to the summarization task, instead, HUPD allows to perform a great variety of tasks. Finally, for what concerns the dimensions, HUPD is absolutely the bigger one. It is followed by BigPatent, which contains about 1/4 of the patents of HUPD. The smallest one is the CMUmine dataset. However, despite its small dimensions, the obtained results are fully satisfying.

Chapter 4

Related works

Text summarization is a natural language processing task that involves automatically generating a concise and informative summary of a longer text document. The goal of text summarization is to distil the most important information and main ideas from a large body of text while minimizing redundancy and maintaining the coherence and overall meaning of the original document [26].

The objectives of text summarization can vary depending on the specific application or domain. For example, in news summarization, the goal may be to provide a brief and informative overview of a breaking news story, while in academic summarization, the goal may be to provide a concise summary of a research paper or technical document for a non-expert audience. In legal or patent summarization, the goal may be to extract key points or legal rulings from a lengthy document for use in a legal or business context.

In recent years, text summarization has been revolutionized by the use of deep learning models, such as neural networks, which have shown impressive results on a range of summarization tasks. These models are typically trained on large-scale datasets of text documents and associated summaries and can learn to capture the most important features and patterns of the text.

There are several typologies of text summarization, based on the techniques and methods used to generate the summary:

1. Extractive summarization: This approach involves selecting the most important sentences or phrases from the original text and assembling them into a summary. Extractive summarization methods typically use statistical or machine learning techniques to identify the most important sentences, such as those that contain high-frequency keywords or appear at the beginning or

end of the document. The advantage of extractive summarization is that it preserves the original wording of the document, which can be useful for legal or technical documents;

2. Abstractive summarization: This approach involves generating a summary that may not be a direct extract from the original text. Abstractive summarization methods typically use natural language generation techniques, such as neural networks, to generate summaries that capture the main ideas and concepts of the original text. Abstractive summarization is more challenging than extractive summarization because it requires understanding the meaning and context of the text, and generating natural-sounding language;

3. Hybrid summarization: This approach combines elements of extractive and abstractive summarization to generate a summary that is both informative and coherent. For example, an extractive summarization algorithm may be used to select the most important sentences from the original document, and an abstractive summarization algorithm may be used to rephrase or generate new sentences to improve the coherence and readability of the summary;

4. Query-based summarization: This approach involves generating a summary that is tailored to a specific user query or information need. Query-based summarization algorithms can use techniques such as keyword extraction, entity recognition, and semantic analysis to identify the most relevant information from the original document and present it in a summary form that is tailored to the user's query.

Overall, text summarization is an important and challenging problem in natural language processing, with many practical applications and research directions. The choice of summarization technique and approach depends on the specific application and the trade-offs between informativeness, coherence, and computational efficiency.

4.1 General-purpose summarization models

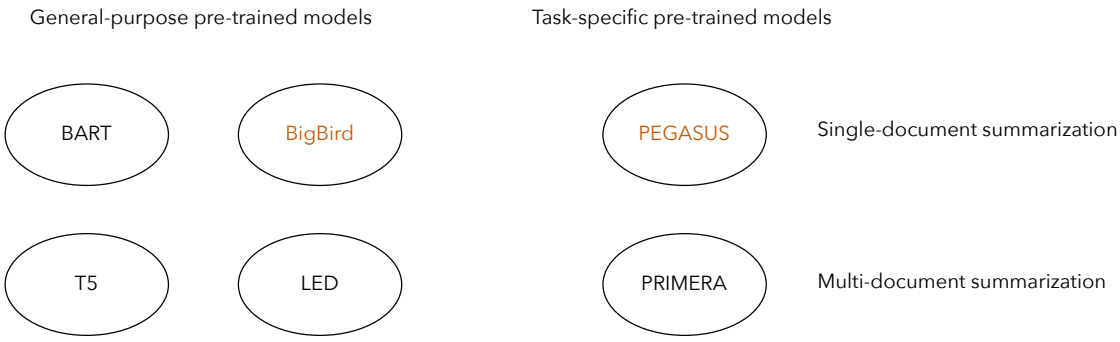


Figure 4.1: This figure is inpired to [27]. It shows the models mostly used in the abstractive summarization field nowadays.

In addition to the already presented abstractive summarization models, PEGASUS [16] (Section 2.2.1) and BigBird [17] (Section 2.2.2), the abstractive summarization field is currently dominated by the models cited in Figure 4.1. A total of six models are described in this work, four out of six are general-purpose pre-trained models, of which Bigbird is part. The two remaining presented models are task-specific pre-trained models, more specifically they are focused on the summarization task. PEGASUS processes single documents, whereas PRIMERA [27] is designed to work with multiple documents at the same time.
In this section BART [28], T5 [29], LED [30] and PRIMERA [27] will be explained. Successively, a particular focus is put on the patent's summarization field. The models and the datasets used will be reported, along with the state-of-the-art results.

4.1.1 BART

BART [28] is a denoising autoencoder implemented employing a bidirectional encoder and a left-to-right autoregressive decoder. BART generalizes BERT [14] for the encoder part, and GPT [15] for the decoder one. Both the modules rely on the transformers' architecture, explained in Section 2.2.1.
BART is built as a sequence-to-sequence model that applies to many end tasks, including abstractive summarization. The model architecture is reported in Figure 4.2
The model makes use of a novel noising function used to corrupt the input. The authors tested several noising functions, but the one that the authors found to be the best is made of two parts:

- a random shuffle of the sentence order;

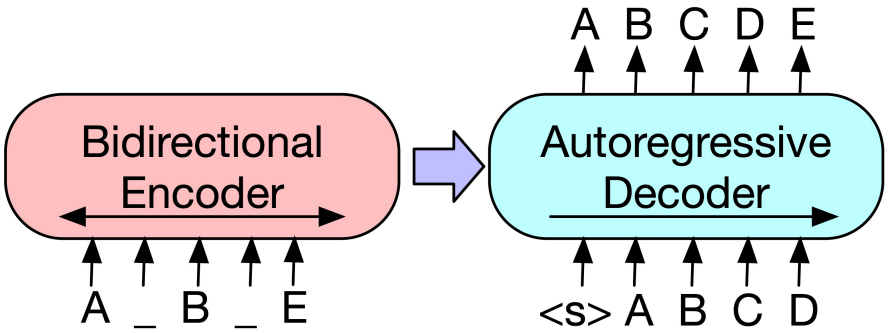


Figure 4.2: BART architecture as explained in [28]. It comprises a bidirectional encoder that takes in input the corrupted text, which is then elaborated by the autoregressive decoder trying to reconstruct the original document.

- a new function that substitutes arbitrary length pieces of text, including length zero, with a single mask token.

This approach generalizes the masking function usually used in BERT, forcing the model to consider long-range transformations of the input and to reason more on the entire sentence length.
The corrupted text is given to the bidirectional encoder, and then, the text representation is passed to the decoder, which learns to correctly reconstruct the original text. More specifically, BART aims to optimize the reconstruction loss, that is the cross-entropy loss, between the decoder's output and the original document.

4.1.2 T-5

Transfer learning is a machine learning method that takes a pre-trained model on data-rich tasks and uses it as a starting point for fine-tuning the model on a downstream task. In that fashion, although the available data for the downstream task is few, it is still possible to reach satisfying performances. The main idea behind the [29] work is to treat each text processing problem as a text-to-text one, therefore, the model always takes an input text and generates a text. This approach allows huge flexibility, going from question answering, to abstractive summarization, or sentiment classification. This approach also allows us to fix the used model, a Transformer, and vary the pre-training objectives or the data sets. Due to these characteristics the model is called *Text-to-Text Transfer Transformer* (T5).
The strategy that allows training a single model on different tasks provides for transforming each problem into a text-to-text one. That is, the model is always fed with a text input, which is then processed. Finally, the model always generates a piece of text. Even with a regression task, the model outputs a text. More

specifically, if the result is a floating-point number, the model's output will be a string containing the computed value.
To tell the model which task is required to perform, some text special tokens are used. For example, for text translation from English to German the model input is: "translate English to German: That is good", the output text will be "Das ist gut". For the summarization task, instead, the text that is desired to summarize is preceded by the "summarize:" token.

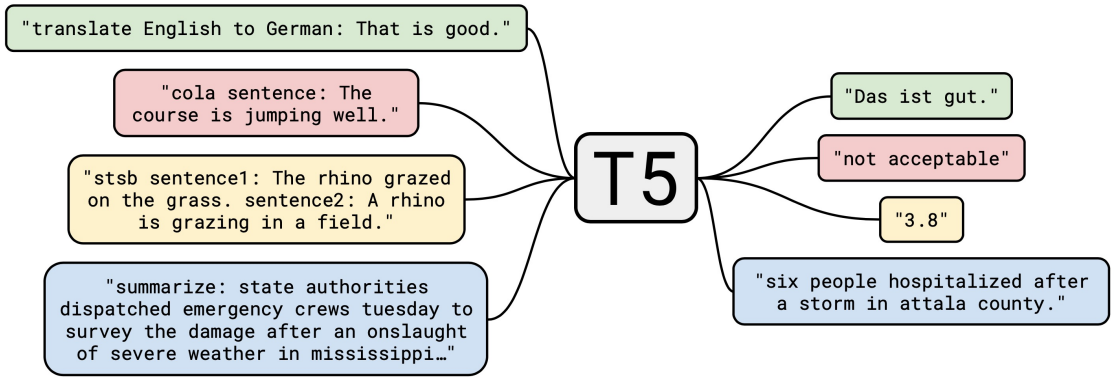


Figure 4.3: A diagram of the text-to-text framework presented in [29]. Each text-processing task is treated as a text-to-text problem. This approach allows using of the same model across different downstream tasks. The model is told to perform a particular downstream task using some special tokens. For this thesis, particular attention should be put on the summarization task, where the input model is preceded by the string "summarize:".

The authors have done the pre-training step on the Transformer model by using a denoising objective function and then fine-tuning the model separately on each downstream task. The used model is such that both the encoder and the decoder are similar in terms of configuration and size to a $\text{BERT}_{\text{BASE}}$ stack [14]. The encoders' attention is usually a full self-attention, the decoders instead use a causal self-attention, that prevents the model from attending to subsequent tokens.
The C4 dataset, an unlabeled dataset presented along with T5, is used for pre-training. With this dataset, the model has to be pre-trained through an objective function that does not require labels. For that reason, the author used a simple denoising objective. More specifically, the function samples and then drops a fixed percentage of tokens in the input sequence. Moreover, consecutive spans of masked tokens are replaced with a single sentinel token.
After extensive analyses, the authors found that, during fine-tuning, updating all the model's parameters leads to higher performances. Therefore, when fine-tuning

the model on each downstream task, all the model's parameters will be updated. This works presents different model sizes, such as the *small* one, used, for instance, in the HUPD presentation [25] to perform the summarization task (Section 3.2.3). This version has an embedding dimensionality of 512, a feed-forward output dimensionality of 2,048, 8 heads of attention, and 6 layers on each encoder and decoder stack.

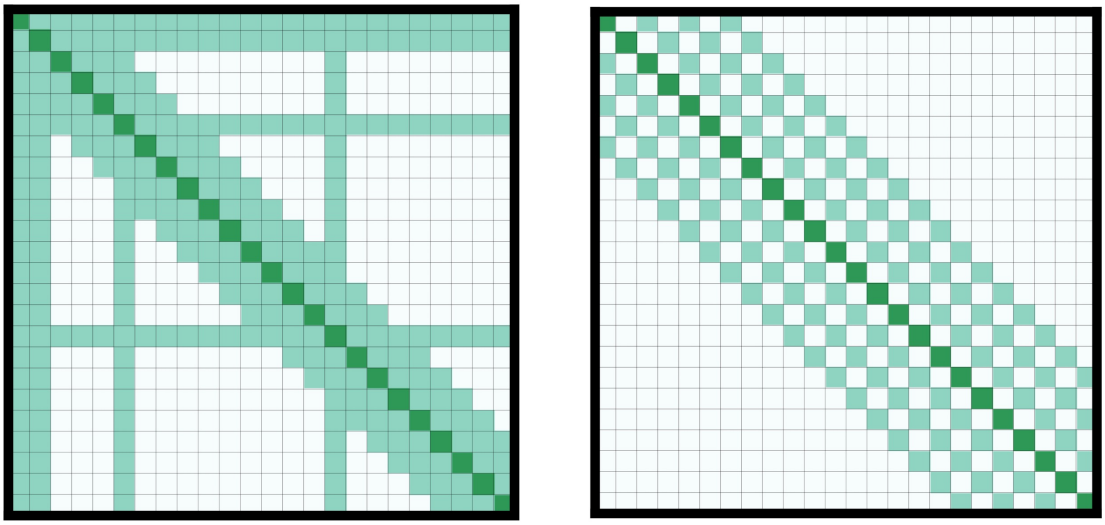
4.1.3 LED

Models based on Transformers [13] have difficulties in processing long-length inputs due to their full self-attention mechanism that scales quadratically with the input length in terms of memory and time complexity. More specifically, its complexity is $\mathbb{O}(n^2)$, where n is the input length.
Longformers [30] are helpful in this field due to their particular attention mechanism that scales linearly with input length, making them well-suited to process long documents. This mechanism provides for a combination of local and global attention. Both attentions are necessary, the local one permits to build of local representations, whereas the global one builds length-long sequence representations.
The Longformer Encoder-Decoder (LED) is a variant of the Longformer architecture that is made of both an encoder and a decoder, rather than by only an encoder. This variant is introduced to be used for seq2seq learning, as the summarization task.
More specifically, the attention used in the encoder is made of two parts:

- a sliding window attention, where the window size is increased in the higher layers. In that way, with a smaller window size for the lower layers, the model can capture local information. At the same time, with bigger window sizes for the higher layers, LED can learn the representation of the entire sequence. The authors also introduced an increasing dilated attention, a windowed attention with several gaps to increase the receptive field, but only in the higher layers of two heads;

- global attention, that is applied to some pre-selected tokens. The important part is that the attention operation has to be symmetric; the pre-selected tokens attend to all the tokens in the input sequence, and, at the same time, all the tokens attend to it.

The attention mechanism is depicted in Figure 4.4. The decoder, instead, uses the standard full self-attention mechanism.
The attention function is the one presented in [13] and reported in Equation 2.6.
The LED architecture and its initialization are the same as BART [28] (Section 4.1.1). The only difference is that the local + global attention mechanism allows increasing the input length to about 16k tokens, against the about 1K of BART.



(a) Local + global attention.

(b) Dilated window attention.

Figure 4.4: Attention mechanism presented in LED [30]. Figure (a) depicts the combination of a local sliding window with the global attention of some pre-selected tokens. Figure (b) represents the dilated window attention used in the higher layers of two attention heads.

4.2 Task-specific summarization models

Task-specific summarization models are models that are designed and trained for a specific application. The task-specific summarization models presented in this thesis are two: PEGASUS (Section 2.2.1) for single document summarization, and PRIMERA for generating multi-document summaries by processing multiple documents simultaneously.

4.2.1 PRIMERA

PRIMERA is a multi-document summarization model presented by [27]. This model uses a newly introduced pre-training objective designed to allow the model to discover how to link and combine data from several documents within a cluster of related texts.

The new strategy relies on the GSG function, firstly introduced in [16], and is called Entity Pyramid. This objective function masks the salient sentences in the whole cluster. The model is led to learn to generate them, discover the most important information across the entire cluster and join them into a single summary. As in GSG, the function selects m informative sentences and substitutes them with masking tokens. The difference lies in the fact that the GSG objective considers a single document, whereas PRIMERA aims to select sentences from several documents to include information coming from different texts of the cluster in the generated summary. The selected sentences contain high-frequency entities. More formally the steps to perform the sentence selection are the following:

1. Perform named entity extraction;
2. Estimate the importance of each entity, the higher the number of the document in which the entity appears, the higher the importance;
3. Remove all the entities that appear in a single document;
4. Select the entities from the top of the pyramid reported in FIGURE 4.5 to the bottom. The entities in the higher level of the pyramid have a higher importance, whereas those in the lower part appear less frequently in the documents;
5. Select the most informative sentences based on their content overlap. More formally, the content overlap is measured utilizing ROUGE [31]. Each sentence score is computed against all the documents except the one to which the considered sentence belongs. That is:

$$Score(s_i) = \sum_{doc_j \in C, s_i \notin doc_j} ROUGE(s_i, doc_j) \tag{4.1}$$

where C is the cluster of related documents.

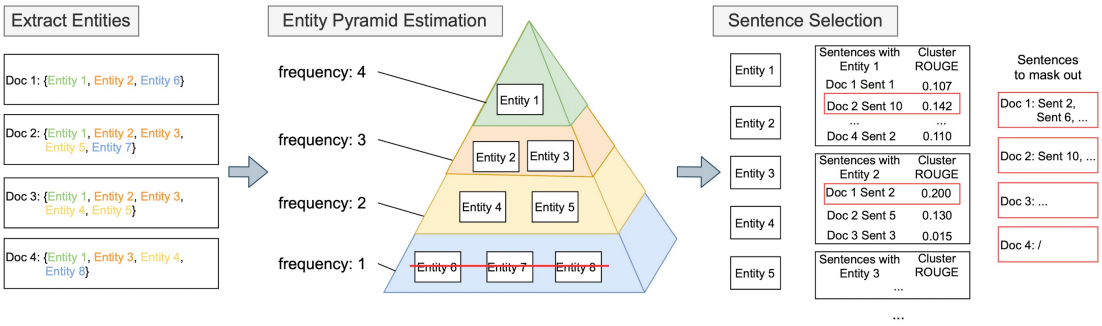


Figure 4.5: Senence extraction steps presente in [27]. (1) Named-entity extraction, (2) Importance estimation of each entity, with subsequent pyramid construction, (3) Remove all the entities that appear in a single document, (4) Find the sentences that contain the informative entities, (5) Select the sentences with a higher sum of ROUGE scores against all the documents to which the sentence does not belong to (Refer to Equation (4.1)).

This strategy encourages the selection of sentences that are informative for many documents in the cluster, rather than favour the exact match with a few documents.

To process multiple documents at the same time the authors simply concatenated the related texts into a single long sequence. Since the input sequence has a notable number of tokens, it is processed with LED (Section 4.1.3) presented in [30], which has a linear complexity in terms of time and memory with the input length. When the text is concatenated, special tokens are added to separate the different documents and to allow the model to consider the document boundaries. These new special tokens are assigned global attention. In such a way they could encapsulate and share information across documents. To better understand the usage of local and global attention in PRIMERA refer to Figure 4.6.
This model can be used also to perform single-document summarization when the input document contains several sections. The maximum length of the input text that the model can handle is 4,096 tokens.

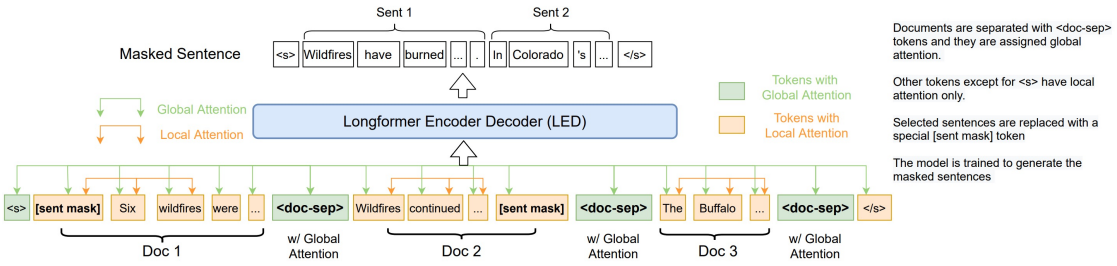


Figure 4.6: Local and golbal attention used is PRIMERA [27].

4.3 Approaches for patent's summarization

Most of the existing datasets for summarization and pre-trained models are related to the news domain, which has different characteristics from the patents one. In particular, patents have a richer discourse structure in their summaries, informative content more evenly distributed in the input text and more abstract summaries.

There are two main approaches used in natural language processing (NLP) for patent summarization: abstractive and hybrid. Abstractive summarization involves generating a summary from scratch by understanding the meaning and context of the input document. This approach is more challenging and requires more advanced NLP techniques such as deep learning and natural language generation. Hybrid approaches instead combine extractive and abstractive techniques to generate a summary that captures both the main ideas and key details of the patent document. In the context of patent summarization, both hybrid and abstractive techniques have been explored, with recent works focusing more on abstractive approaches.

4.3.1 SentRewriting

In June 2019, the authors of the well-known patent dataset, BIGPATENT, observed that existing abstractive summarization models produced lower ROUGE scores on patents compared to those obtained in the news domain. This was due to the models excessively repeating irrelevant discourse elements in the generated summaries.

BIGPATENT contains only two fields: the detailed patent description as input, and the golden summary section as the target output. Therefore, the only summarization task that could be performed is to elaborate the description to obtain a text that resembles the summary.

In 2018 the authors of SentRewriting [32] is a research paper that explores a novel approach to abstractive text summarization using a combination of reinforcement learning and sentence rewriting. The proposed method focuses on improving the efficiency and effectiveness of abstractive summarization by selecting and rewriting the most informative sentences in the source document. This is achieved through a process of reinforcement learning, where a model is trained to select and rewrite sentences that maximize a summary quality score. In particular, they trained a Pointer Network to extract sentences recurrently. For what concers the reinforcement learning model, at each step t the model is rewarded with the ROUGE-L_{F1} score between the generated sentence and the ground truth sentence.

More formally, given g the abstractor function and d_{j_t} the extracted sentence by the Pointer Network at step t, $g(d_{j_t})$ is the generated sentence at the considered step. Given also s_t the ground truth sentence at step t, the reward at each step can be computed as:

$$r(t+1) = \text{ROUGE-L}_{\text{F1}}(g(d_{j_t}), s_t) \tag{4.2}$$

A sketch of the main steps performed is reported in Figure 4.7.

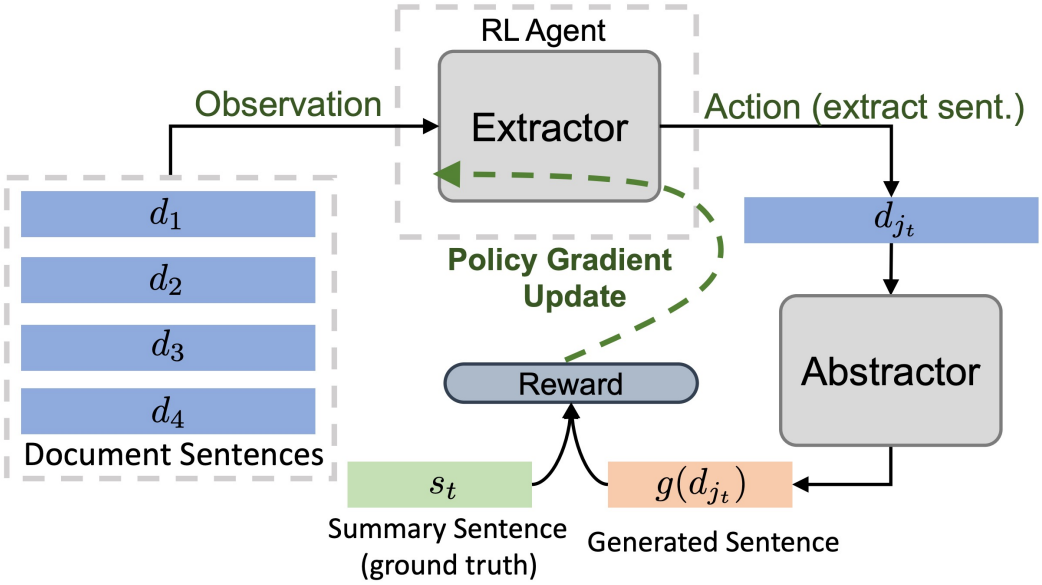


Figure 4.7: SentRewriting approach presented in [32].

The results of the study show that this approach outperforms existing methods. The obtained ROUGE scores are reported in Table 4.1.

Model	ROUGE-1	ROUGE-2	ROUGE-L
SentRewriting	37.12	11.87	32.45

Table 4.1: ROUGE scores obtained using SentRewriting on BigPatent, therefore using the description as input text and the summary as ground truth.

4.3.2 Hierarchical Seq2Seq Sentence Pointer + Transformer Language Model

In 2020 Subramanian et al. [33] showed the benefits of mixing an extractive approach with an abstractive one also in the patent domain. In particular, they used

a Hierarchical Seq2Seq Sentence Pointer to extract the most informative sentences and then they performed the abstractive step with the Transformer Language Model presented in [13], conditioned on the extracted sentences. To perform the abstractive summarization part the authors used a transformer language model that is trained from scratch on the appropriately formatted data. The language model they used is GPT-2 [34].

The authors organized the training data in such a way that the selected sentences were concatenated after the input text and then given to the model. In such a way they were able to model the joint distribution of the document and summary during training and then sampled from the conditional distribution of the summary given document at inference time. The authors' experiments revealed that utilizing the ground truth extracted sentences during the training phase, and the model-extracted sentences during the inference phase resulted in better performance compared to using only the model-extracted sentences throughout the summarization process.

They also introduced a special token to identify the start of the summary and used it at inference time as a signal for the model to start generating the summary.

The results obtained applying the model of BigPatent, therefore taking in input the description trying to generate the abstract are reported in Table 4.2.

Model	ROUGE-1	ROUGE-2	ROUGE-L
TLM	36.41	11.38	30.88
TLM + E	**38.65**	**12.31**	**34.09**

Table 4.2: ROUGE scores obtained by Subramanian et al. [33] on BigPatent [7]. TLM stands for Transformer Language Model to be intended without a previous extraction step. The TLM + E indicates the extractive step is applied before the summary generation, which is then used to condition the Transformer Language Model on relevant information.

4.3.3 PEGASUS

In July 2020 the authors of PEGASUS [16], described in Section 2.2.1, tested their models on the patent summarization downstream task using BIGPATENT. In particular, they tested three models: $\text{PEGASUS}_{\text{BASE}}$, $\text{PEGASUS}_{\text{LARGE}}$ pretrained on C4 [29] that contains web-extracted texts and $\text{PEGASUS}_{\text{LARGE}}$ pretrained on Hugenews, a dataset the authors collected containing newslike articles. Since

the dataset used is BigPatent, the summarization task is performed by using the description as the source of information, trying to generate the summary section.

The obtained scores are reported in Table 4.3.

Model	ROUGE-1	ROUGE-2	ROUGE-L
$\text{PEGASUS}_{\text{BASE}}$	43.55	20.43	31.80
$\text{PEGASUS}_{\text{LARGE}}$ (C4)	**53.63**	**33.16**	**42.25**
$\text{PEGASUS}_{\text{LARGE}}$ (Hugenews)	53.41	32.89	42.07

Table 4.3: ROUGE scores obtained by [16] on BIGPATENT [7] with patent descriptions as input and summaries as ground truth. The models tested are $\text{PEGASUS}_{\text{BASE}}$, $\text{PEGASUS}_{\text{LARGE}}$ pretrained on C4, $\text{PEGASUS}_{\text{LARGE}}$ pretrained on Hugenews.

As one can notice from Table 4.3 the best scores are achiecieved by $\text{PEGASUS}_{\text{LARGE}}$ pretrained on C4.

Another work that makes use of PEGASUS is the one presented with the CMUmine dataset, described in Section 3.2.2.
All the aforementioned works are focused on the generation of a patent section that has to be created by the inventor, the CMUmine dataset [3] instead focuses on the generation of the first claim, a task that is up to the IP attorney. Typically, the attorney has to deeply understand the invention disclosure and then formulate the claims. BIGPATENT cannot be used to address this problem, since it contains only the description and the summary sections. CMUmine instead, containing sections such as background, abstract and claims, in addition to detailed description and summary, not only allows to address the problem but also to study which section is the most informative one to generate the first claim. The first claim generation is addressed as an abstractive summarization problem.
In particular, the authors tested two models: PointGenerator [35] and PEGASUS [16]. They used the training data set to train the model and the validation set and to finetune the hyperparameters.

The results using PEGASUS, the summary as the source of information and the first claim as ground truth are reported in Table 4.4.

Model	ROUGE-1	ROUGE-2	ROUGE-L
PEGASUS	75.97	64.47	70.54

Table 4.4: ROUGE scores obtained by [3] using PEGASUS on the CMUmine dataset are presented. The input text is the summary section, and the golden truth is the first claim.

4.3.4 Point Generator

The authors of CMUmine [3] in addition to PEGASUS also tested Point Generator [35] on their newly introduced dataset.

The fundamental architecture of the Pointer Generator is a sequence-to-sequence attention model, commonly known as an encoder-decoder model. Point Generator network is capable of both generating words from a fixed vocabulary and copying words by pointing to them. The generation probability p_{gen} is computed at each time step t using the context vector and decoder states, as illustrated in Figure 4.8. This probability acts as a soft switch that enables the model to choose between generating a word from the vocabulary distribution P_{vocab} or copying a word from the input sequence by sampling from the attention distribution.

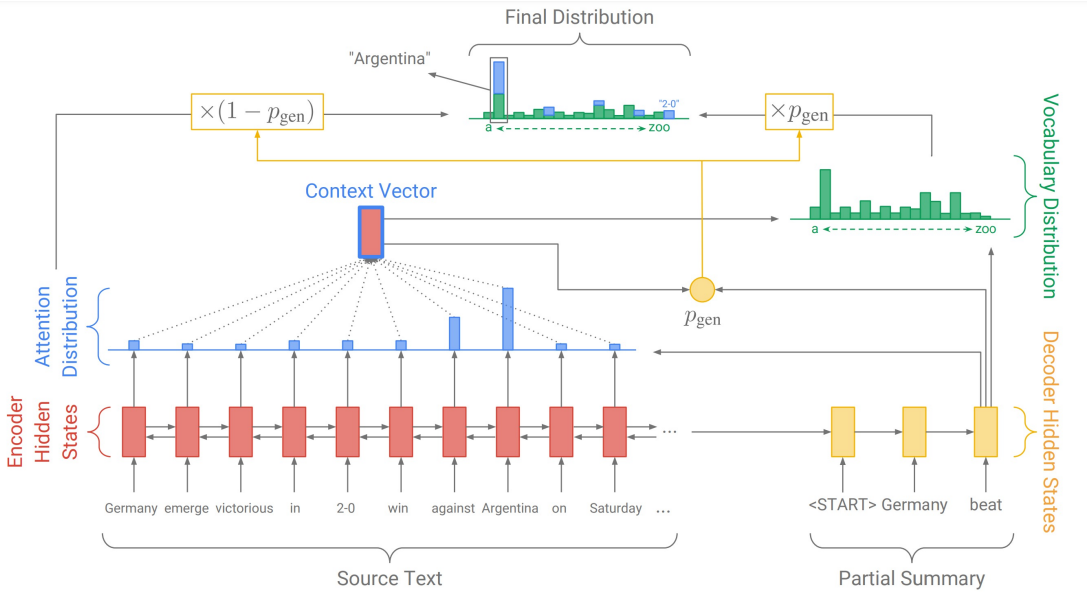


Figure 4.8: Point Generator network presented in [35].

The summarization task performed is to take the summary as input text and predict the first claim. The ROUGE scores obtained using Point Generator are reported in Table 4.5.

Model	ROUGE-1	ROUGE-2	ROUGE-L
Point Generator	65.51	52.95	59.95

Table 4.5: ROUGE scores obtained by [3] using Point Generator on the CMUmine dataset are reported. The input model is the summary section, and the golden summary is the first claim.

4.3.5 T5 small

In 2022, the HUPD dataset was introduced, which differs from previous patent datasets by containing all sections of the patents, providing the opportunity to tackle different summarization tasks. In particular, the authors of the dataset aimed to generate the abstract section from two distinct sections, namely the description and the claims. To achieve this, they utilized the T5 model, specifically the small version discussed in Section 4.1.2. The results of the experiments are presented in Table 4.6.

Input section	ROUGE-1	ROUGE-2	ROUGE-L
Descritpion	62.87	47.20	54.36
Claims	**69.00**	**53.82**	**59.88**

Table 4.6: ROUGE scores obtained by [25] using the small version of T5 on HUPD. Two different inputs are compared, the description and the claims, whereas the golden truth is the patent abstract.

The results demonstrate how the input section could affect the text summarization performance.

4.4 Comparison of summarization results in patent domain

This section is devoted to the direct comparison of the aforementioned models used for patent summarization.
According to Table 4.7 best model to generate the summary starting from the description, using BigPatent, is $\text{PEGASUS}_{\text{LARGE}}$ pre-trained on C4. For what concerns the summarization that aims to reconstruct the abstract, using HUPD, it is possible to notice that, using T5 small, the most informative input section is the one that contains the claims. Finally, for the task that takes in input the summary section, trying to predict the first claim, using CMUmine the best model is PEGASUS.

Model	Input → Output	ROUGE-1	ROUGE-2	ROUGE-L
SentRewriting	Description → Summary	37.12	11.87	32.45
TML	Description → Summary	36.41	11.38	30.88
TML + Extraction	Description → Summary	38.65	12.31	34.09
$\text{PEGASUS}_{\text{BASE}}$	Description → Summary	43.55	20.43	31.80
$\text{PEGASUS}_{\text{LARGE}}$ (C4)	Description → Summary	**53.63**	**33.16**	**42.25**
$\text{PEGASUS}_{\text{LARGE}}$ (Hugenews)	Description → Summary	53.41	32.89	42.07
T5 (small)	Description → Abstract	62.87	47.20	54.36
	Claims → Abstract	**69.00**	**53.82**	**59.88**
$\text{PEGASUS}_{\text{LARGE}}$	Summary → First Claim	**75.97**	**64.47**	**70.54**
Point Generator	Summary → First Claim	65.51	52.95	59.95

Table 4.7: Comparison of the ROUGE scores obtained by the previous works in patent summarization. The results are divided on the basis of the ground truth section.

This thesis is focused on the first claim generation, indeed task claim generation is a highly important part to keep up with the speed of new patent filing. Moreover, this work makes use of the PEGASUS model since the results shown in Table 4.7 point out that it achieves very good performances on the considered task. For all the experiments the CMUmine dataset is used.

Chapter 5

Methodology

The objective of this thesis is to scrutinize and extend the results obtained by the CMUmine authors [3] (Section 3.2.2). In particular, the study focuses on two main research questions:

- which patent sections are the most effective for generating the first claim;
- how does the length of the input text impact the performance of the summarization models.

This research could have significant implications for the legal and innovation communities by improving the accuracy and efficiency of automated patent claim generation.

Seven different input texts are analyzed, including single sections like abstract, background, description, and summary, as well as combinations of two sections. The boundaries of each section are highlighted using special tokens to help the models recognize the semantic content. More details of the input creation are reported in Section 5.2.
To investigate the impact of context width, two models are compared: PEGASUS and BigBird-PEGASUS, both based on the transformer architecture but with different attention mechanisms, that lead to different maximum input length. More specifically, PEGASUS's maximum input length is 1,024, whereas the BigBird-PEGASUS's one allows input texts up to 4,096 tokens. More details are reported in Section 5.3.
The experiments are performed on the CMUmine dataset, which contains around 300,000 patents.

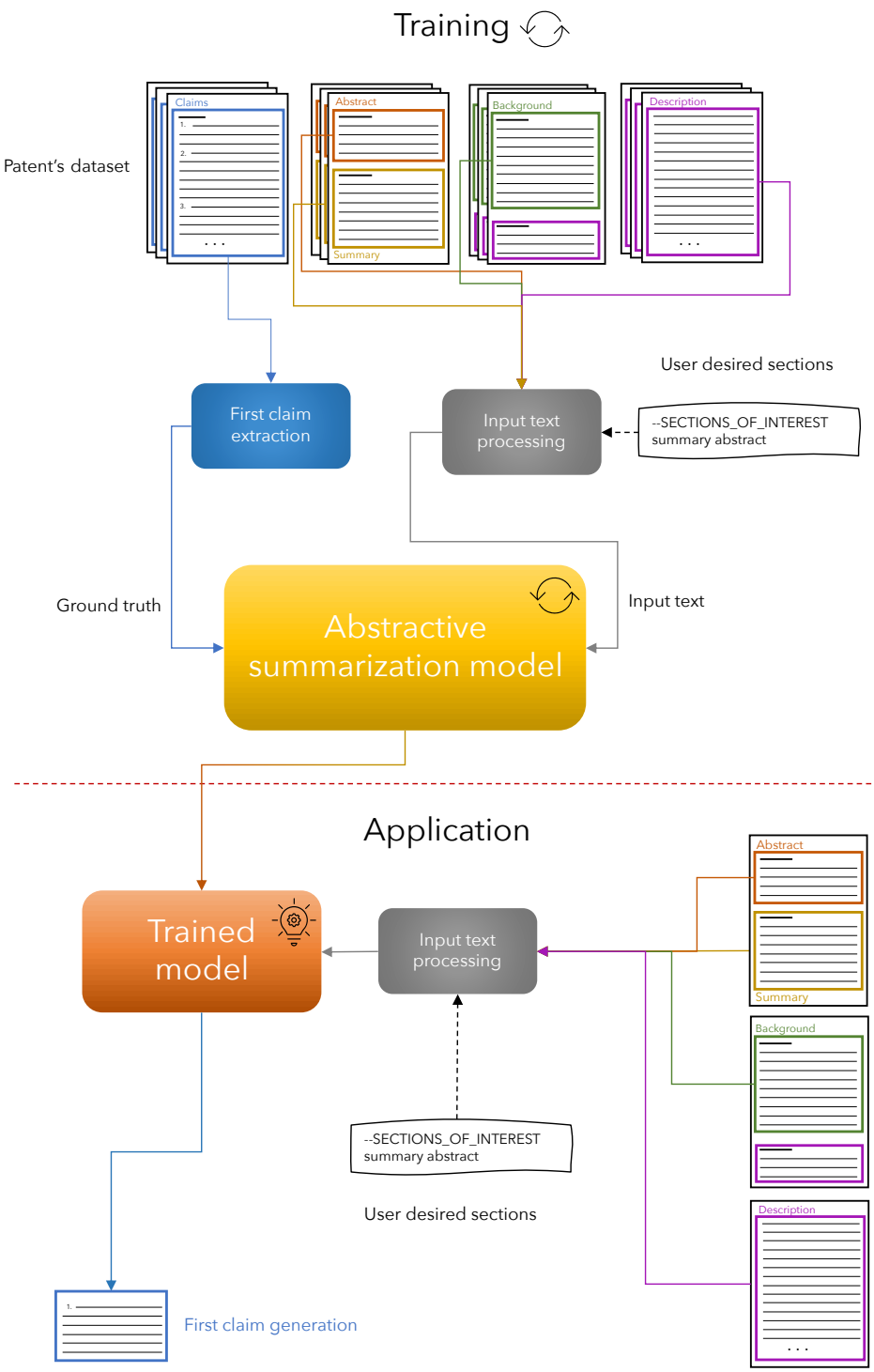


Figure 5.1: Sketch of the main steps of this thesis.

5.1 Datasets preparation

5.1.1 CMUmine

The CMUmine dataset's [3] authors made all the data available at the following Google Drive link: `https://drive.google.com/drive/u/0/folders/1J4sAcM_21G39VuZT1jv6RqLTEM_UngWS` (last accessed: January 2023).
It is divided into two main directories, one containing data related to 2005, and the other to 2006. Each directory is divided into three subfolders, one containing the training data, one for the data used for evaluation, and the last one for the data used to test the model.

This data format is not a common form, therefore, to make this thesis modular, the data of the two years are firstly concatenated and then transformed into a single Huggingface `DatasetDict`[1]. In such a way, dealing with a common data format, this thesis can be easily adapted to many other datasets provided by Huggingface, for instance, the HUPD dataset[2] [3], doing only a few modifications. The created `DatasetDict` has three splits, one for the training, one for the evaluation and the other for the test. Each entry has seven fields whose names are: patent_number, abstract, background, summary, detailed_description, firstclaim and claims. See Section 3.2.2 for more details.

To apply this work to different patent datasets, one has to do the following things:

- to identify and isolate the first claim if not already present as a separate field;
- to rename the corresponding section to match the exact names used for the CMUmine dataset;
- if not already present create three splits to train, validate, and test the model.

After a study of the dataset, it has been found that about 34% of documents in each of the training, validation and test splits are not provided with the description section. The situation is reported in Figure 3.1. One has decided to maintain all the documents for two main reasons: the first one is that removing all these documents can lead to a performance decrease due to a relevant amount of data loss. The second one is that among the seven different experiments performed, and

[1] `https://huggingface.co/docs/datasets/package_reference/main_classes` (last accessed: January 2023)

[2] `https://huggingface.co/datasets/HUPD/hupd` (last accessed: January 2023)

explained in the following section, only one uses the description section. Therefore, the lack of a section affects only one out of the seven results obtained for each tested model. The issue is addressed by using an empty string as the description of the documents that are not provided with it.
The prepared version of the CMUmine dataset is available at `https://politoit-my.sharepoint.com/:f:/r/personal/s283832_studenti_polito_it/Documents/CMUmine_datasetdict_complete?csf=1&web=1&e=aih2Of`.

5.1.2 HUPD

A second dataset is prepared: HUPD[2]. This dataset is provided by the Huggingface dataset hub. Due to its sheer size, the dataset is only prepared and not used due to the time limits of the computations imposed by the available resources but needed to perform even a single epoch of the faster model on the whole dataset.

The HUPD dataset has a total of 34 fields, but this thesis is focused only on a small amount of them. For this reason, and for limiting memory usage, the dataset is pruned. In particular, the only preserved fields are patent_number, abstract, background, summary, description and claims. This dataset contains all the claims of each patent, but since this work is focused only on the first one, it is necessary to check whether each patent contains the first claim and then isolate it for future computations. It is done by splitting the claims section on the '. 2' string. The motivations behind this string are that each claim is composed of a single sentence, therefore, a full stop succeeded by an empty space signals the end of a claim. The number '2' is motivated by the fact that the focus is posed exactly on claim number '1', and not the first claim reported in the claims section.
The retrieved first claim is saved in the firstclaim field of the patent entry.

Moreover, to maintain the same code used to process the CMUmine dataset, all the field names are modified to match their corresponding names in the CMUmine dataset. For instance, the description field originally present in HUPD, becomes the detailed-description field, as in CMUmine. During this operation 8.2% of the total documents have been removed because the claims section is not provided with claim number one, which has been cancelled. More specifically, in these cases, the section starts with a string like: '1.-5. (Cancelled)'. Although these documents have been removed, the number of remaining documents was still considerable, indeed more than 4 million documents are preserved.

The dataset was originally divided into two single splits, training and testing. The training split contains all the patents filed between January 2004 and December 2016, whereas the test set the document filed between January 2017 and December

2018. This thesis also needs the validation one. It has been created by doing a random sample from the training set of the same number of documents contained in the test set.

5.2 Input text creation

During training time the sections are prepared to be fed to the model. In particular, an argument passed to the model signals which are the sections of interest that it has to consider.

This thesis makes use of some new special tokens that are able to identify the boundaries of each considered section. They also allow the model to recognize the semantic content of each section. The new tokens added to the tokenizer are eight. Those that are concatenated at the beginning of the section, start with the letter B, whereas those used to signal the end of the section, start with the letter E. More specifically the used new special tokens are:

- *[B-ABS]* and *[E-ABS]* used to signal the beginning and the end of the abstract section;
- *[B-SUMM]* and *[E-SUMM]* used to identify the boundaries of the summary;
- *[B-BACK]* and *[E-BACK]* employed to recognize the start and the end of the background section;
- *[B-DESC]* and *[E-DESC]* utilized to point out where the description starts and where it ends.

Before the training operation a `torch.Dataset` class is created to correcly handle the dataset.
In this class, the target text is set to have a maximum output dimension of 275 tokens. This is done because the 99th percentile of the first claim length is 275 tokens. Refer to Table 3.1 for more details. In that way, the overwhelming majority of the first claims are fully contained in the maximum output length that the model is able to generate in output.
For what concerns the input text, its maximum length depends on the models used. In this work, it varies from 1,024 tokens when using PEGASUS, to 4,096 when the adopted model is BigBird-PEGASUS.
During training each input text is created in the `__getitem__` function. Therefore, when a document is retrieved, the corresponding input text is created. Each desired section is concatenated at the beginning and at the end with special tokens related

to the considered section. After each section is processed in that fashion, all the desired sections with the corresponding new special tokens are concatenated.
The class receives in input the tokenizer that already contains the new special tokens used to signal the boundaries of each section. In such a way during the encoding phase, the tokenizer is able to recognize and correctly codify them.

The tested inputs are seven, all the single sections plus three combinations of several sections. For all the section's average and 99th percentile lengths refer to Table 3.1.
In particular, the four single-section input texts are:

- Abstract: the 99^{th} percentile of its length lies both in the maximum input length of PEGASUS (1,024) and BigBird-PEGASUS (4,096);
- Summary: its average length lies abundantly in the maximum input length of both the models, whereas the 99th percentile, with 1,440 tokens, exceeds the PEGASUS input limit;
- Background: its situation is similar to the one of the summary section, therefore, its mean length can be easily processed by both the models, whereas the 99th percentile of about 2660 tokens signals that only BigBird-PEGASUS will be able not to lose information from many patents;
- Description: its 99^{th} percentile largely exceeds both the maximum input length with its about 19k tokens. For what concerns the average length exceeds the PEGASUS limit, but satisfies the BigBird-PEGASUS one.

The three test input combinations made by two different sections are:

- Summary + Abstract;
- Summary + Background;
- Background + Abstract.

The considered input texts are depicted in Figure 5.2.
This work does consider neither combinations of three sections nor combinations that include the description section due to the excessive length the input will reach, leading to a certain input truncation, thus, not providing any performance improvement.

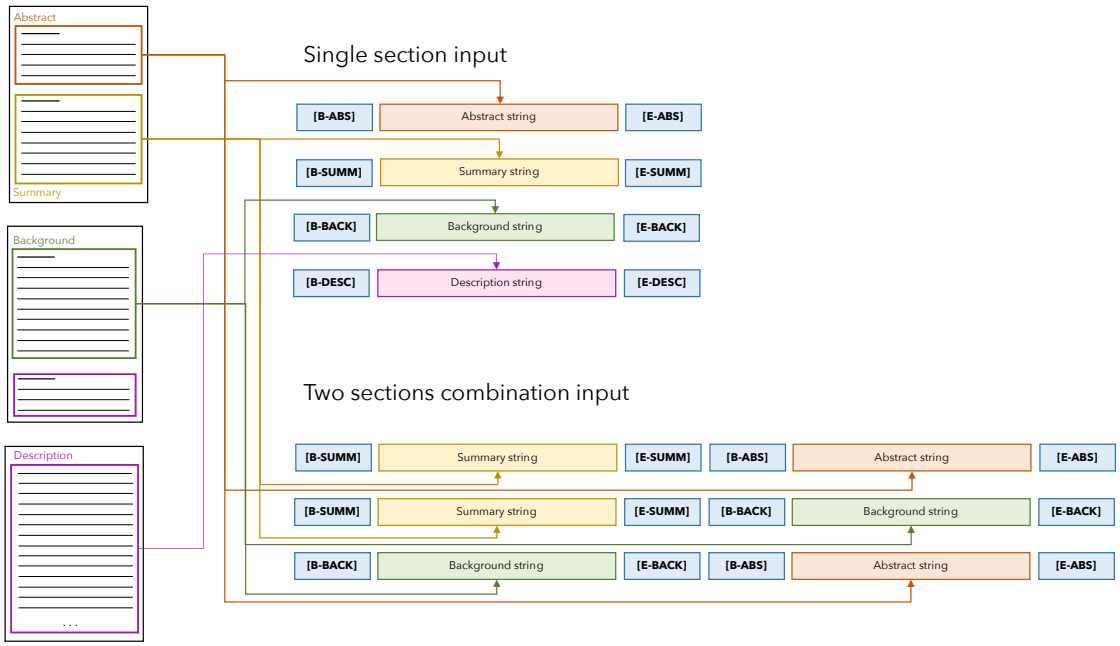


Figure 5.2: Graphical representation of the input text creation. Each desired section is firstly concatenated with the correct input tokens, at the beginning and at the end of the section. If there are multiple sections desired, the newly obtained strings are concatenated to obtain the correct input text.

5.3 Used models

The models used in this thesis are PEGASUS [16] and BigBird-PEGASUS [17], explained in Section 2.2.1 and Section 2.2.2 respectively.

In order to process documents in the patent domain both the models are taken from the Huggingface hub already pretrained on BigPatent [7]. More specifically, the PEGASUS model present at link `https://huggingface.co/google/pegasus-big_patent` (last accessed: January 2023) is used, whereas the adopted BigBird-PEGASUS model can be found at `https://huggingface.co/google/bigbird-pegasus-large-bigpatent` (last accessed: January 2023). In such a way, the used models have already dealt with the patent language, which is quite different from the conventional English language. It contains much more legal terms, specific language and terms related to a particular art. In particular, they have already faced the description language as input, therefore, the models have learned the information on the general language typical of the patent domain. The training scheme representation is reported in Figure 5.3.

The maximum input length of PEGASUS is set to its maximum, that is 1,024 tokens. The maximum input length of BigBird, instead, is set to 4,096, which is its maximum triable input length. This difference in the input lengths allows this work to inspect whether an increase in context length affects the achieved performances and in which way.

This thesis extensively uses the Transformers API to download and train the pretrained models.
In particular, the `PreTrainedModel` class is used to load the pretrained model configurations downloaded from the Huggigface repository. The `PreTrainedTokenizer` class implements the commonly used methods to encode the strings that a model has to process in model inputs and, in this work, is used to load the pretrained tokenizers provided by the Huggingface library. The new special tokens are added through the `add_special_tokens` method, after that, the token embedding matrix is updated to be able to deal with the new special tokens. This is achieved by employing the `resize_token_embeddings` method.
The `compute_metrics` function used during training is defined in such a way that ROUGE-1, ROUGE-2, ROUGE-L and ROUGE-Lsum are computed for each generated first claim against the golden one.
Both the used models exploit the `DataCollatorForSeq2Seq`. Data collators are the objects in charge of creating the batches using a list of the elements present in the dataset. The training arguments are always defined by means of the `Seq2SeqTrainingArguments` class, which allows the model to reach a deep level

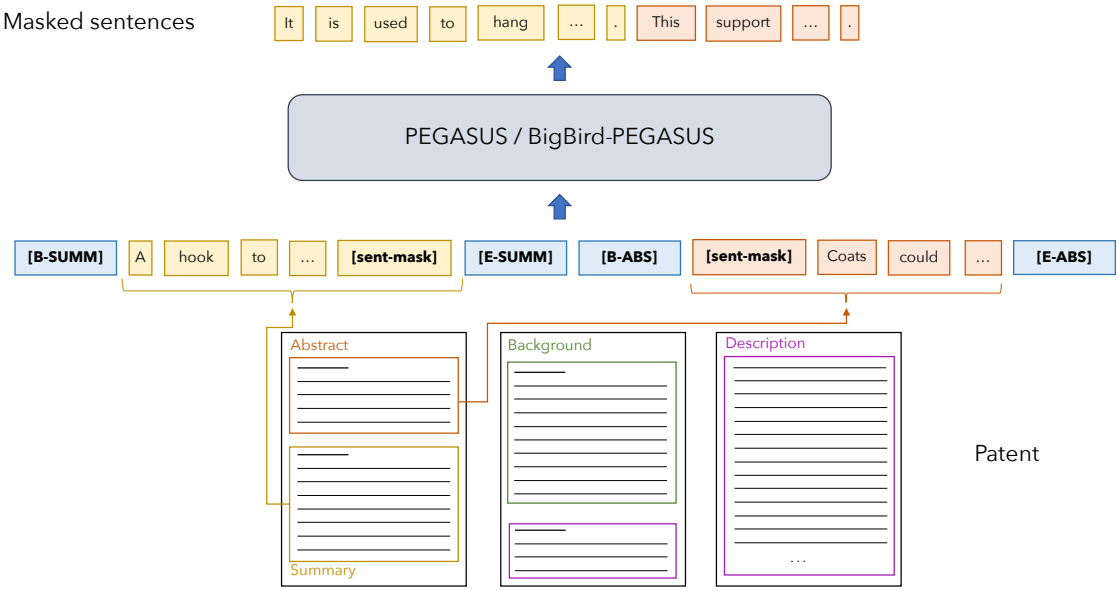


Figure 5.3: The strategy used during training is reported in this figure. The model is fed with the desired input sections. For instance, the sections of interest in this example are the summary and the abstract. The sections are concatenated firstly with the corresponding special tokens and then with one another. Then the GSG masking function typical of PEGASUS [16] is applied. The adopted model is then trained to predict the masked sentences.

of customization during training. After the training arguments are defined the `Seq2SeqTrainer` class is used to perform the training using PyTorch.
When the model is required to be trained for more epochs, the best model to be preserved is chosen based on the ROUGE-2 score it achieves on the validation set.

5.4 Input parameters used for training

To perform the desired computations the user has to define some important parameters. In addition to the standard parameters such as the batch size, the learning rate, and the number of epochs, some parameters allow the user to choose some important settings. The most important parameters to be set are:

- –MODEL_NAME_OR_PATH: this parameter allows the user to define the model to use. It can be retrieved locally or from the Huggingface library of pre-trained models. This thesis uses two variants: *google/pegasus-big_patent* if the desired model is PEGASUS pretrained on BigPatent, and *google/bigbird-pegasus-large-bigpatent*, if the used model is BigBird-PEGASUS, pretrained on BigPatent;
- –MAX_INPUT_LENGTH: this value sets the maximum input length that the model has to process. This parameter is passed to the `Dataset` class that creates the input data format. See section 5.2 for more details. It has to be set to 1,024 if the used model is PEGASUS and to 4,096 if the desired model is BigBird;
- –DATASET_NAME: the current version of the code allows only two possibilities, *CMUmine* to use the CMUmine dataset prepared as a `DatasetDict`, as explained in Section 5.1.1, and *HUPD* to exploit the prepared version of the dataset as described in Section 5.1.2;
- –SECTIONS_OF_INTEREST: this parameter allows the definition of the sections to be given in input to the model. The possible section names are : *abstract*, *summary*, *background*, *detailed_description*. If the combination of multiple sections is desired, it suffices to separate the desired section names with a single space. In that way, there are no constraints on the number of sections to be used, thus, if the user wants to use more than two sections it is possible.

5.5 At test time

During the test phase, two metrics are computed: ROUGE (Section 6.1.1) and BERTscore (Section 6.1.2). The first one is used mainly to compare the obtained performances with the previous works. Given that it endorses the syntactic similarity rather than the semantic one, also the BERTScore is computed.
Some parameters are added to allow a great level of customization for the user. In particular, in addition to the same parameters introduced in Section 5.4, some parameters are used:

- –SAVE_DICTIONARY: when this parameter is used a `.txt` file containing a dictionary with all the predictions with the respective ground truth is saved. This document allows us to perform the predictions using the GPU a single time, further metrics can be computed exploiting only the CPU, having a time reduction;
- –PRINT_SAMPLES: it is used to print the predictions with their ground truths in the log file along with the computed metrics. The idea is to allow the user to inspect forthwith the quality of the first claim;
- –EVALUATE_ONLY: when this parameter is used the model retrieves the predictions previously computed and only computes ROUGE and BERTScore. This is the setting that allows using only the CPU.

Chapter 6

Results

This section is dedicated to discussing the results obtained using the proposed methodology. In particular, in Section 6.2 the experimental configurations that led to the results presented in the successive section are reported. To reduce the GPU memory usage the gradient checkpointing technique is applied. The general rule is that this approach slows down the computations by about 20%. To regain some computational speed, almost without loss in performance, the mixed precision approach is preferred to the full precision one.

6.1 Metrics

The generated summaries are evaluated through two well-known summarization metrics: ROUGE (Section 6.1.1) and BERTScore (Section 6.1.2).

6.1.1 ROUGE

The ROUGE score [31] is presented as a set of measures that permits an automatic assessment of the quality of a computer-generated summary concerning a set of ideal reference summaries written by humans. This metric counts the number of n-grams overlapping between the candidate summary and the reference ones. ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation.

ROUGE-N ROUGE-N is an n-gram recall between the candidate summaries and the set of ideal ones generated by humans. More specifically, let R_S be the set of reference summaries. The ROUGE-N score is obtained as follows:

$$\text{ROUGE-N} = \frac{\sum_{S \in R_S} \sum_{\text{n-gram} \in S} count_{match}(\text{n-gram})}{\sum_{S \in R_S} \sum_{\text{n-gram} \in S} count(\text{n-gram})} \tag{6.1}$$

where $count_{match}$ is the number of co-occurring n-grams in the candidate and reference summary and n is the length of the n-gram. Since at the denominator there is the sum of all the n-grams of the reference summaries, it is clear there the ROUGE score is a recall-oriented measure.

ROUGE-L ROUGE-L considers Longest Common Subsequence. More formally, given 2 sequences X and Y, the longest common subsequence is the common subsequence with the higher length. The idea is that the longer the longest common subsequence the more similar the candidate and the ideal summary

The ROUGE score is a popular metric in summarization, indeed, a higher ROUGE means a bigger overlap of n-grams between reference and candidate summary.
This metric, unfortunately, has some main drawbacks. First of all, it gives the same importance to all the n-grams, although some terms should be given a higher relevance, whereas some terms should have a lower importance. In the second instance, this score does not evaluate the summaries' fluency and does not take into account synonyms. This leads to higher ROUGE scores for extractive summarization, whereas abstractive summarization, which tends to maintain the same meaning using different words may lead to lower scores even if a human judgement considers it a good summary.

6.1.2 BERTScore

BERTScore [36] is a metric that allows the automatic evaluation of generated texts. It measures the similarity of each token in the candidate text with each token in the reference one leveraging on pre-trained BERT [14] contextual embeddings and not on exact matches. More specifically, this score measures the similarity of two sentences as the sum of the cosine similarity of their token embeddings. Using this technique, the BERTScore is able to successfully match paraphrases and capture distant dependencies and word order.
Given a reference sentence and a candidate one, the algorithm computes the BERT contextual embedding of each token, evaluates the pairwise cosine similarity and sums the values properly weighted by the *idf* frequency. More formally, the steps are the following:

1. Given the reference sentence $x = \langle x_1, ...x_k \rangle$ and the candidate one $\hat{x} = \langle \hat{x}_1, ...\hat{x}_l \rangle$ compute each token contextualized embedding by means of BERT;

2. Compute pairwise cosine similarity. The usual equation to compute the cosine

similarity is

$$\frac{x_i^T \hat{x}_j}{\|x_i\| \, \|\hat{x}_j\|} \tag{6.2}$$

where x_i is a reference sentenc and $\hat{x}_j$. However, the authors decided to use pre-normalized vectors in order to simplify computations and reduce the cosine similarity to a mere inner product $x_i^T \hat{x}_j$;

3. The equations used to compute the BERTScore are precision (6.4), recall (6.3) and F1 (6.5). The final score is given by the F1 measure;

$$R_{BERT} = \frac{1}{|x|} \sum_{x_i \in x} \underset{\hat{x}_j \in \hat{x}}{max} \, x_i^T \hat{x}_j \tag{6.3}$$

$$P_{BERT} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \underset{x_i \in x}{max} \, x_i^T \hat{x}_j \tag{6.4}$$

$$F_{BERT} = \frac{2P_{BERT} \cdot R_{BERT}}{P_{BERT} + R_{BERT}} \tag{6.5}$$

4. Each token is associated with its *idf* given that rarer works can be more indicative of similarity rather than more common ones. This value is used to adjust the precision and recall scores;

5. Since the author used pre-normalized vectors the scores lays in the range [-1,1] as the cosine similarity. For ease of readability, the score is rescaled between 0 and 1.

6.2 Experimental setup

Both the used models are large models that make heavy usage of the GPU memory, running the risk to run out of memory while working. Indeed, to compute the gradients during the backward pass usually, all the activations of the forward pass are saved to be then used to compute the gradients. This can lead to a memory overload, still implying a shorter training time. A possible solution is to never save the forward pass activations but recompute them when needed during the backward pass. This strategy needs much lesser memory but adds a significant computation load, slowing down training.
This thesis adopts a compromise approach: gradient checkpointing. This approach saves some intermediate forward activations. The saved nodes of the computational graph are called checkpoints and can be manually provided or automatically selected. This technique leads to training time and memory usage that is in between those needed by the two aforementioned approaches.
To enable gradient checkpointing during training time it is only needed to set to `True` the `gradient_checkpointing` flag and pass it to the `TrainingArguments`. In this work, it is only necessary to add the –GRADIENT_CHECKPOINTING parameter in the training script.

The general rule is that using gradient checkpointing slows down training by about 20%, therefore, to regain some computational speed an additional approach is applied: mixed precision training. The idea behind this approach is that not every variable needs to be stored with the 32-bit floating point precision, also called full precision. The main advantage comes from using the 16-bit floating point precision, or half-precision, for all the activations during the forward pass and also during gradients computations at the backward pass. Although the gradients are computed in half-precision they are converted to full-precision for the optimization steps. Since the model uses both half (16-bit) precision and full (32-bit) precision, this training approach is called mixed precision training. This precision reduction leads to almost no effects on the model performance. To enable the mixed precision training with both models it is only necessary to set the `fp16` flag to `True`. To do that in this thesis is only necessary to add the –FP16 parameter in the training script. During the training of PEGASUS, in the cases in which it receives as input the abstract or the description the loss suddenly goes to `NaN`. This problem is solved by restoring the full precision in these two cases, with a subsequent reduction of the batch size.

The used optimizer for both the models is the `AdamW`, which is the Adam optimizer with the weight decay. Adam, which stays for Adaptive Movement estimation,

is an algorithm that allows the model to perform big learning steps when the gradients do not change much, whereas, when the gradients vary rapidly the learning steps are small. In the implementation of Adam the regularization term, that is the weight decay, is added before the batch gradient computation, thus, the gradients computed keep also track of the regularization factor and not only of the weights as it should be. The solution comes with AdamW, in this algorithm the weight decay is performed only after the gradient of the batches is computed. See [37] for more details.

For what concerns the batch sizes of the models, the first observation to do is that the batch size of PEGASUS can be much higher than the maximum allowed by BigBird-PEGASUS. It is due to the different maximum input lengths of the two models, 1,024 and 4,096, respectively. Therefore, BigBird-PEGASUS allowing a wider context requires a lower batch size. In particular, PEGASUS allows a maximum batch size during training of 32, whereas BigBird-PEGASUS is trained with a batch size of 12, which are the highest allowed to respect the memory limits. During the evaluation, the allowed maximum batch size is lower. In particular, a batch size of 10 is used during the PEGASUS evaluation. BigBird-PEGASUS is instead evaluated with a batch size of 6. During the test both the models use a batch size of 6.

During training, PEGASUS has a learning rate of 1e-4, whereas BigBird-PEGASUS of 1e-5. The learning rate of 1e-4 has been tested on BigBird-PEGASUS, but it turned out to be too high.

During the evaluation, the ROUGE scores are computed. When the model is trained for more epochs, the best model is chosen based on the highest achieved ROUGE-2 score.

For the experiments the computational resources are offered by HPC@POLITO (`http://www.hpc.polito.it`). More specifically, the experiments are conducted on an NVidia Tesla V100 with 32 GB of CUDA memory and 5120 CUDA cores.

The implemented code will be available at the following URL:
`https://github.com/MorenoSara/Transformers-based-Abstractive-Summarization-for-the-Generation-of-Patent-Claims`

For what concerns the number of epochs, PEGASUS can be trained for a maximum of 5 epochs, whereas BigBird reaches a maximum of a single epoch. These constraints are imposed by the HPC@POLITO's computation time limits of 5 days. To perform a fair comparison of the two models they are trained for a single epoch.

The hyperparameter used is reported in Table 6.1.

Hyperparameters	PEGASUS	BigBird
Learning rate	1e-4	1e-5
Training batch size	32	12
Evaluation batch size	10	6
#epochs	5	1
Max input length	1,024	4,096
Max output length	275	275
Mixed precision (fp16)	yes (except Abstract and Description)	yes
Gradient checkpointing	yes	yes

Table 6.1: Hyperparameters used to train PEGASUS and BigBird

6.3 Obtained results

This section reports the results obtained by applying the two models each of seven different text inputs. To perform a fair comparison both models are trained for a single epoch.

6.3.1 PEGASUS results

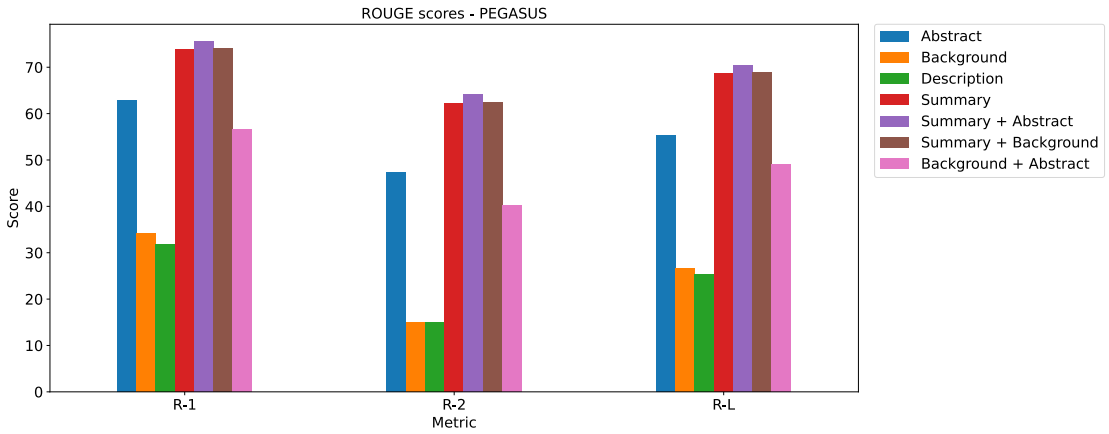
The PEGASUS ROUGE score results are shown in Figrue 6.1, whereas the precise values are reported in Table 6.2

Model input	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-Lsum
Abstract (no fp16)	62.81	47.25	55.38	58.71
Background	34.17	14.96	26.56	30.56
Description (no fp16)	31.9	14.97	25.26	28.57
Summary	73.84	62.11	68.66	70.86
Summary + Abstract	**75.49**	**64.1**	**70.44**	**72.58**
Summary + Background	74.11	62.38	68.92	71.13
Background + Abstract	56.57	40.24	49.06	52.53

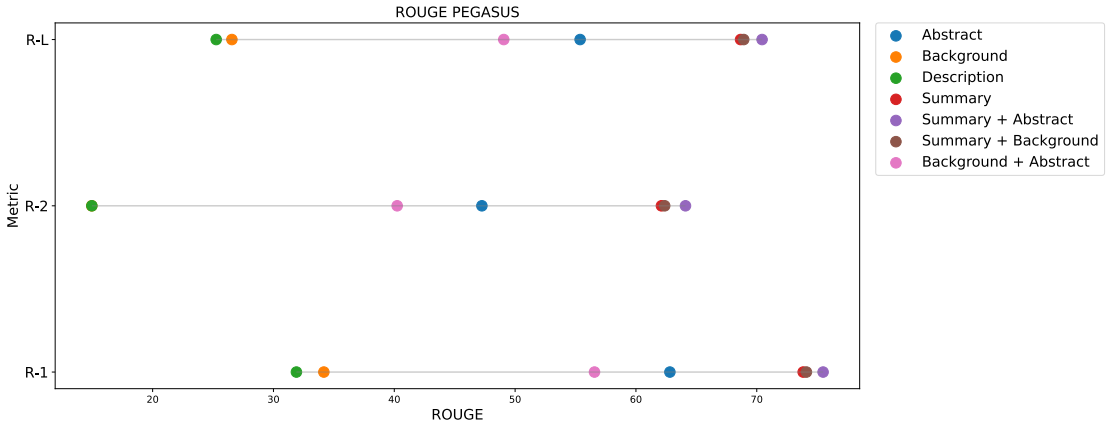
Table 6.2: ROUGE scores obtained with PEGASUS, trained for a single epoch, with all the tested inputs. The highest ROUGE-1, ROUGE-2, ROUGE-L and ROUGE-Lsum scores are achieved by the summary and abstract input combination.

As Figure 6.1 the model that receives in input the combination of summary and abstract sections reaches the best scores in each ROUGE metric.
In general, the single section that allows one to reach the best scores is the summary, followed by the abstract.
Instead, for what concerns the input made of two sections, the best scores, as previously said, are achieved by the summary and abstract combination, followed by scores slightly lower by the summary and background combination.
As it is possible to notice, every time the summary is contained, both as a single section and in a combination, the ROUGE scores are higher than those obtained with the other sections. The only combination that produces a valuable increase in the performance compared to the one of the only summary section is the combination that considers also the abstract, the second single section for performance.

It is interesting to highlight that although ROUGE favours the syntactic similarities to the semantic ones, as explained in Section 6.1.1, the achieved ROUGE



(a) Global view



(b) Detailed view

Figure 6.1: Graphical representation of the ROUGE scores reported in Table 6.2, achieved with PEGASUS. The highest performances for all the metrics are achieved by the combination of summary and abstract, followed by the combination of summary and background, followed, in turn, by the summary section and then by the abstract one. It is possible to notice that the background, the description and the combination of background and abstract lead to lower scores.

scores are significantly high, especially considering that the task performed is an abstractive summarization one. Indeed, for instance, the 75.49 ROUGE-1 score obtained with summary plus abstract is a remarkable value.

For what concerns the BERTScore, the results are shown in Figure 6.2. Since the range in which the values lie is limited, between about 82% and 93%, the performance differences are not highly visible in Figure 6.2a. For this reason, Figure

6.2b reports a detailed view that allows one to inspect the relative position of each input section. To better interpret the figures the detailed scores are reported in Table 6.3. The same trend described for the ROUGE scores is valid also for the BERTScore ones. According to both BERTScore precision, recall and F1, the best input is the combination of summary and abstract, followed by the combination of summary and background. They are, in turn, followed by the summary section and then by the abstract.

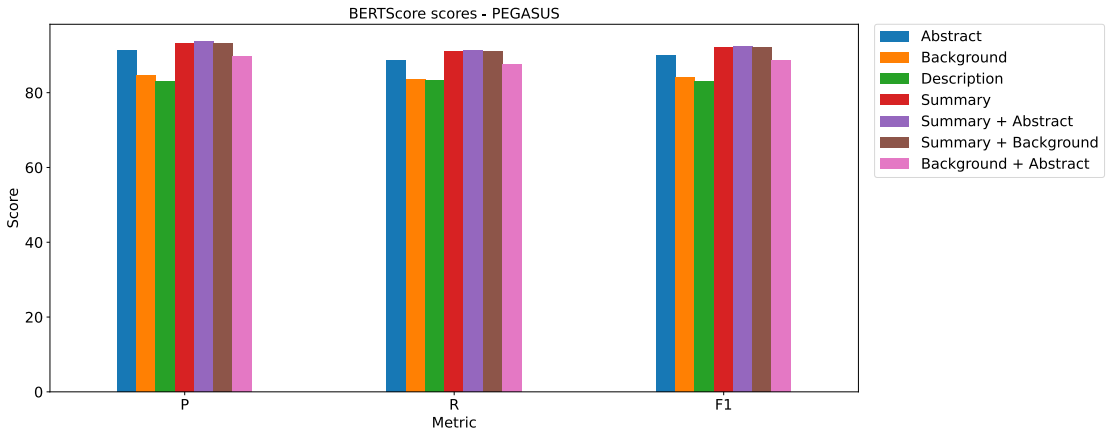
Model input	BERTScore P	BERTScore R	BERTScore F1
Abstract (no fp16)	91.34	88.72	89.99
Background	84.58	83.5	83.99
Description (no fp16)	82.92	83.31	83.06
Summary	93.23	91.09	92.13
Summary + Abstract	**93.61**	**91.39**	**92.47**
Summary + Background	93.28	91.14	92.18
Background + Abstract	89.76	87.59	88.63

Table 6.3: BERTScore precision, recall and F1 results obtained using PEGASUS trained for a single epoch. Here are reported the scores obtained with each input tested. The best scores are achieved by the summary and abstract combination.

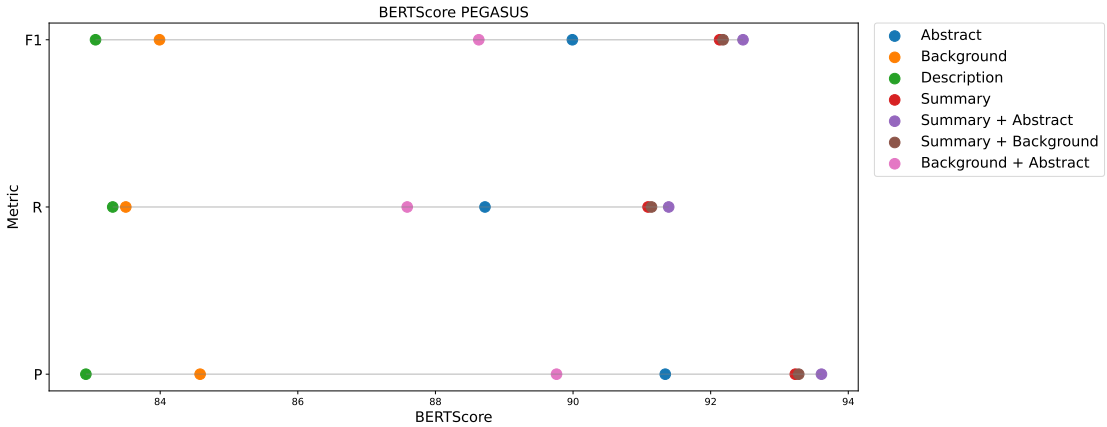
6.3.2 BigBird-PEGASUS results

In this section the results obtained by BigBird after a single training epoch are reported.

Figure 6.3 depicts the rouge scores reported in Table 6.4. As it is possible to see, the three inputs that lead to the highest scores are first the combination of summary and abstract, then, almost with the same score, the summary section and the combination of summary and background. As for PEGASUS, the highest scores are achieved every time the summary is given to the model, both as a single section and in combination with another section. In addition, one can notice that the abstract and the combination of abstract and background reach almost the same performance, differently from what happens with PEGASUS. This is probably due to the low performances obtained by BigBird with the background input, which therefore contains little important information for the claim generation. In general, the performances obtained with the description and the background are the lowest. Another difference with the PEGASUS ROUGE scores is that with BigBird the performance improvement between the only summary and the combination of



(a) Global view



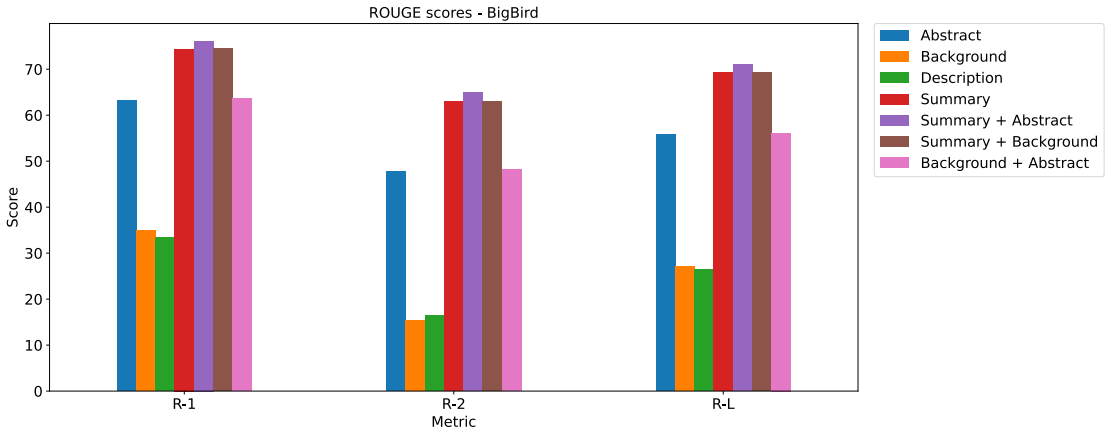
(b) Detailed view

Figure 6.2: Graphical representation of the BERTScore scores reported in Table 6.3, achieved with PEGASUS. The performance difference is not highly visible in Figure (a) due to the limited range in which the scores lie, i.e. between about 82% and 93%. For this reason, a detailed view that allows one to better inspect the relative scores of the different inputs is depicted in Figure (b). The best scores are obtained with the combination of summary and abstract, then with the combination of summary and background, which is in turn followed by the summary section and then by the abstract one. Background, description and the combination of background and abstract lead to lower scores.

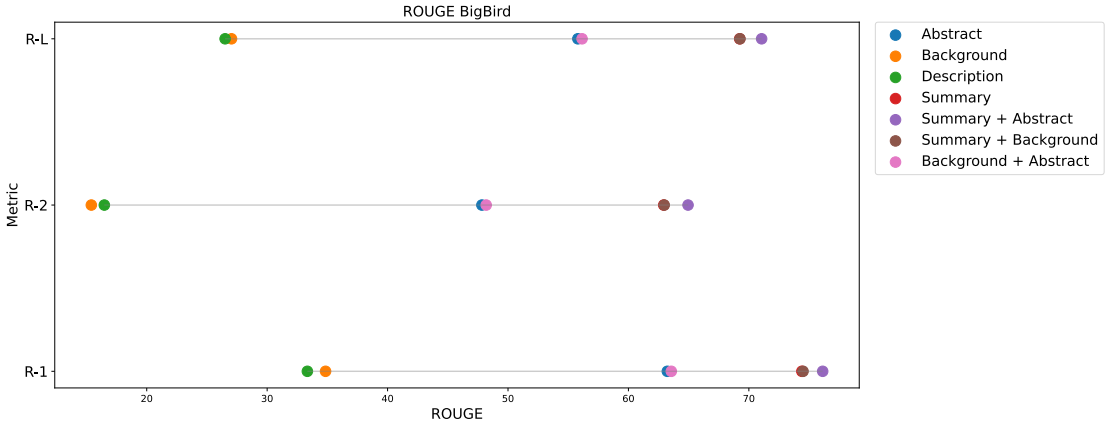
summary and background is really small.

Also using BigBird the highest ROUGE scores are remarkable, especially for an

abstractive summarization task. Indeed, the 76.13 ROUGE-1 score obtained with summary plus abstract is a high value.



(a) Global view



(b) Detailed view

Figure 6.3: Graphical representation of the ROUGE scores achieved by BigBird and reported in Table 6.4. The highest performances obtained for all the ROUGE-1, ROUGE-2 and ROUGE-L scores are reached by the summary and abstract combination, at the second place there is the combination of summary and background, followed by the summary section. Then, with lower values, there are the abstract and the summary that reaches almost the same score. The least informative input sections are the background and the description.

For what concerns, the BERTScore results achieved using BigBird are shown in Figure 6.4. Also in this case, the detailed values are presented in Table 6.5. As it is

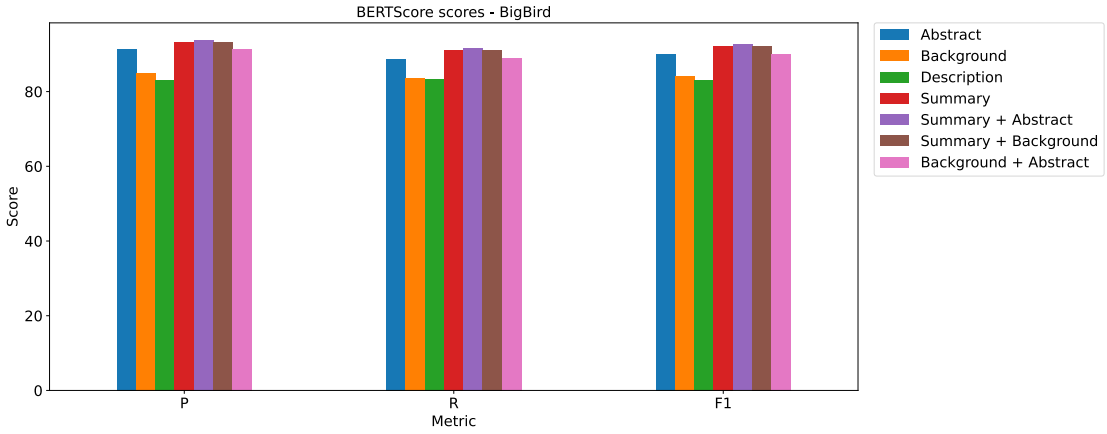
Model input	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-Lsum
Abstract	63.23	47.83	55.8	59.13
Background	34.85	15.4	27.04	31.15
Description	33.34	16.48	26.51	29.91
Summary	74.4	62.93	69.24	71.46
Summary + Abstract	**76.13**	**64.95**	**71.06**	**73.22**
Summary + Background	74.5	62.97	69.26	71.51
Background + Abstract	63.56	48.19	56.15	59.44

Table 6.4: BigBird ROUGE scores obtained after a single training epoch for each input tested. The best scores are reached by the summary and abstract input combination.

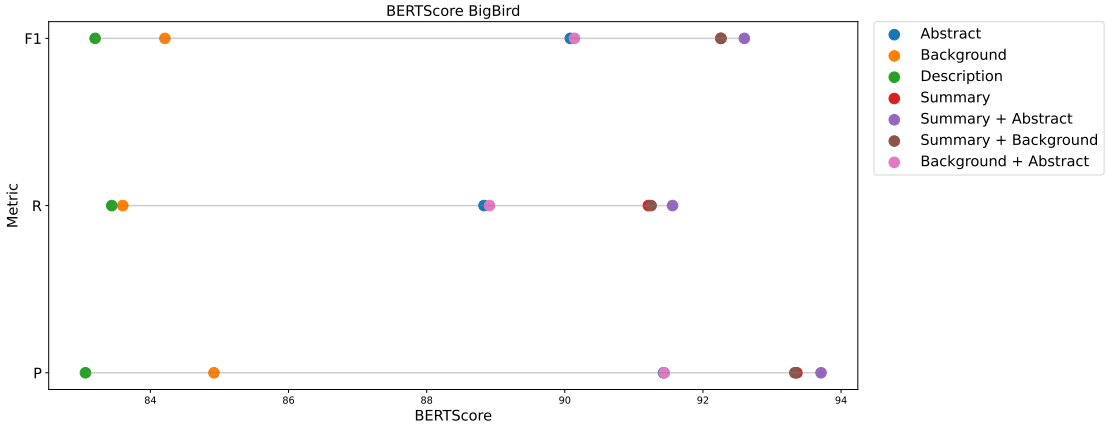
possible to notice, the same trend identified for the ROUGE score is present also for the BERTScore values. The best input is the combination of summary and abstract, followed by a summary and a combination of summary and background, which have almost the same scores. They are followed in turn by the abstract section and the background and abstract combination, with almost the same score. The results highlighy that the least informative sections analyzed are the background and the description.

Model input	BERTScore P	BERTScore R	BERTScore F1
Abstract	91.43	88.83	90.08
Background	84.92	83.6	84.21
Description	83.06	83.44	83.2
Summary	93.36	91.21	92.26
Summary + Abstract	**93.71**	**91.56**	**92.6**
Summary + Background	93.33	91.25	92.26
Background + Abstract	91.44	88.91	90.14

Table 6.5: BigBird-PEGASUS BERTScore results obtained after one training epoch for all the input tested. The best input is a combination of summary and abstract.



(a) Global view



(b) Detailed view

Figure 6.4: Graphical representation of the BERTScore values achieved by BigBird and reported in Table 6.5. Figure (a) reports the global situation of the scores, whereas Figure (b) allows better inspect the relative position of each input text. The best input is the summary and abstract combination, followed almost equally by the summary single section input and the combination of summary and background. They are in turn followed equally by the abstract input and by the background and abstract combination. The least informative input sections turned out to be the background and the description.

6.4 Training times

This section is dedicated to comparing the training times necessary to train PEGASUS and BigBird for a single epoch, with BigBird being trained with the half-precision approach for each input section, while PEGASUS requires full precision when fed with the abstract or description.

As shown in Table 6.6, the results indicate that PEGASUS requires significantly less time than BigBird, about one-third of the time.

Additionally, the reported times demonstrate the efficacy of the mixed precision approach in accelerating the processing time. PEGASUS trained with the abstract and description sections necessitates the full precision approach. This led to a significant increase in the training time, exceeding 10 hours, and results in approximately 30 hours for each epoch, as opposed to the average of 20 hours for other sections.

Model input	PEGASUS	BigBird
Abstract	30:04:27 (fp)	67:30:30
Background	21:49:09	70:21:48
Description	32:20:29 (fp)	71:08:44
Summary	18:57:15	68:15:08
Summary + Abstract	18:45:39	77:54:11
Summary + Background	26:16:56	70:28:57
Background + Abstract	19:27:28	67:13:13

Table 6.6: This table reports the training times necessary for training PEGASUS and BigBird for a single epoch. All the models are trained with the half-precision approach, except for the abstract and the description when using PEGASUS. These cases are denoted with (fp), which stands for full precision.

6.5 Direct comparison of the two models

This section is dedicated to a direct comparison of the two models.
The following figures report only the ROUGE-1 score and the BERTScore-F1 since the trend that can be identified is the same for all the other metrics.
Figure 6.5 shows the comparison of the ROUGE-1 scores obtained with PEGASUS (blue) and BigBird (orange).

As it is possible to notice the performance obtained with each input section is improved when using BigBird. In particular, the highest improvement is achieved for the combination of the background and abstract sections. This same trend appears also in the BERTScore-F1 as presented in Figure 6.6.
The motivation could be that the 99th percentile of the number of tokens in the background section is about 2600 tokens, as reported in Table 3.1. This means that using PEGASUS, which handles texts of 1,024 tokens at maximum, many background sections are truncated. Therefore, concatenating the abstract section to it leads to longer input texts, again truncated. The consequence of it is that PEGASUS performances of the background and summary combination are somehow in between the scores obtained with only the background and only the abstract. BigBird instead, with the capability of handling input text up to 4,096 tokens, can deal with both the background and the combination of background and abstract. This fact leads to obtain scores with the background and abstract combination that are very similar to those obtained with the only abstract in input.

Probably, the summary and background combination is not affected by the same problem since the summary, which leads to the highest scores, is the first part of the input section, then followed by the background. Indeed, the performance is only slightly better than those obtained with the only summary section in the input.

Another important point to highlight is the training time of the two models since they are really different. Indeed, PEGAUS for a single training epoch with respective evaluation epoch on the CMUmine dataset (Section 3.2.2) needs about 22 hours. BigBird instead needs much more time. More specifically, for a single training epoch with a respective evaluation epoch, always with the CMUmine dataset, it needs about 3 days. Since the time limit offered by the computational resources is of 5 days BigBird cannot be trained for more than one epoch. Therefore, to be consistent and do a fair comparison, its results are compared to PRGASUS trained with a single epoch too.
In light of this, if the amount of data is remarkable and the time factor is an important key, PEGASUS could be the better option. It leads to performances only slightly lower than those obtained with BigBird but it implies almost 1/3 of

the training time.

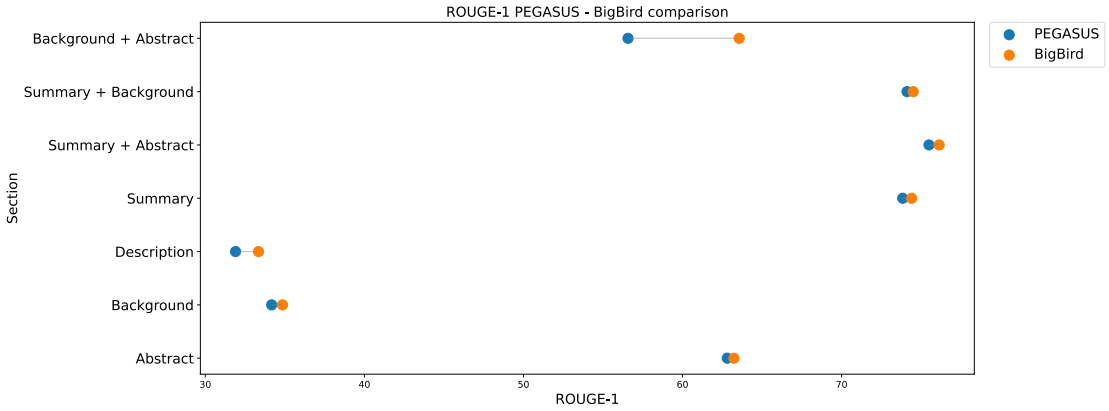


Figure 6.5: This figure reports the performance improvement obtained using BigBird as compared to PEGASUS. Using each input text the performance is always improved, in particular in the case of the combination of background and summary. Since all ROUGE-1, ROUGE-2 and ROUGE-L show the same trend only one of them is reported, in this case, ROUGE-1.

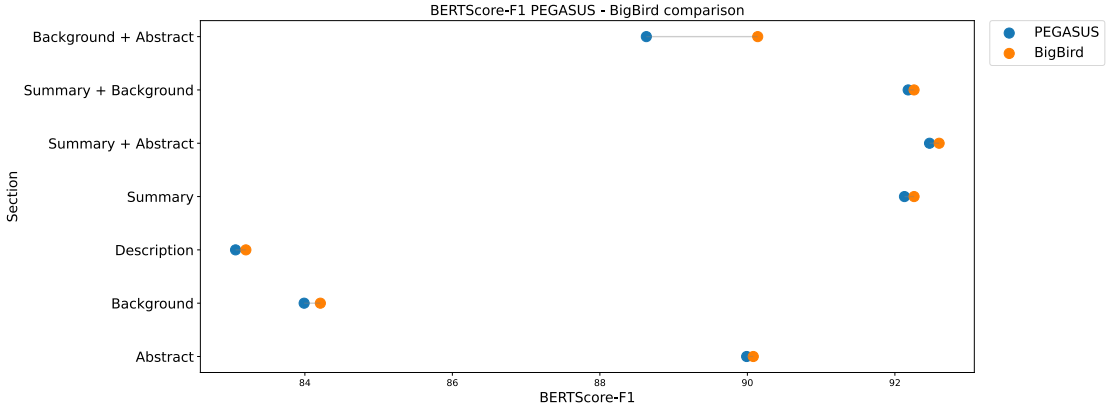


Figure 6.6: This figure shows the performance variations of the BERTScore-F1 results using PEGASUS or BigBird. For all the input texts BigBird always improves the performance. The highest improvement is obtained for the background and abstract combination. Since both precision, recall and F1 present the same trend, only the F1 measure is reported.

6.6 Qualitative analysis with a textual example

This section presents a random, non-cherry-picked textual example to allow a qualitative evaluation of the model performance. For ease of interpretation, the differences with the ground truth are highlighted in bold. The results are ordered starting from the one with the highest score, summary plus abstract, to the one with the lowest score, description.

6.6.1 PEGASUS

In this section, a first claim generated with PEGASUS is reported. It is presented for the input texts analysed. At first, the texts are presented, and then some qualitative analyses are explained.

Ground truth *"1. A bicycle wheel driving device comprising: hub axle; hub shell rotatably arranged around the hub axle, has a hub flange on an outside circumference thereof, and has a housing space inside; and a motor disposed inside the hub shell and configured to rotate the hub shell with respect to the hub axle; and roller brake device at least partially arranged within the hub shell and extending outside of the hub shell."*

Summary + Abstract *"1. A bicycle wheel driving device comprising:* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the hub shell with respect to the hub axle; and a roller brake* ***device arranged*** *within the hub shell* ***to the outside of the housing space****."*

Summary + Background *"1. A bicycle wheel driving device comprising:* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the textbfhousing space and configured to rotate the hub shell with respect to the hub axle; and a roller brake* ***device arranged*** *within the hub shell* ***to the outside of the housing space****."*

Summary *"1. A bicycle wheel driving device comprising:* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the textbfhousing space and configured to rotate the hub shell with*

respect to the hub axle; and a roller brake ***device arranged*** *within the hub shell* ***to the outside of the housing space****."*

Abstract *"1. A* ***motor-equipped hub*** *comprising:* ***a*** *hub axle;* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***the hub shell having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space* ***on the*** *inside* ***thereof****;* ***a*** *motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the hub shell with respect to the hub axle; and a roller brake* ***device arranged*** *within the hub shell* ***to the outside of the housing space****."*

Background + Abstract *"1. A* ***motor-equipped hub*** *comprising:* ***a*** *hub axle;* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***the hub shell having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space* ***on the*** *inside* ***thereof****;* ***a*** *motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the hub shell with respect to the hub axle; and a roller brake* ***device arranged*** *within the hub shell* ***to the outside of the housing space****."*

Background *"1.* ***A wheel driving device configured to drive a wheel of a bicycle using an electric motor, the*** *wheel driving device comprising:* ***a hub having a*** *hub axle* ***and a*** *hub shell* ***that covers the hub axle; a stationary casing fixed to the hub axle; a brake drum fixed to a cylindrical part formed so as to protrude from a bearing supported portion of the hub shell; and a plurality of brake shoes provided on the stationary casing.****"*

Description *1. "****A method of manufacturing a semiconductor device, comprising the steps of: (a) providing a semiconductor substrate; (b) forming a first insulating layer on the semiconductor substrate; (c) forming a first conductive layer on the first insulating layer; (d) forming a second insulating layer on the first conductive layer; (e) forming a second conductive layer on the second insulating layer; (f) forming a third insulating layer on the second conductive layer; (g) forming a third conductive layer on the third insulating layer; (h) forming a fourth insulating layer on the third conductive layer; (i) forming a fourth conductive layer on the fourth insulating layer; (j) forming a fifth conductive layer on the fourth conductive layer; (k) forming a fifth conductive layer on the fifth conductive layer; (l) forming a sixth conductive layer on the fifth conductive layer; (m) forming a sixth conductive layer on the sixth conductive layer; (n) forming a seventh conductive layer on the seventh conductive layer; (p) forming a seventh***

conductive layer on the seventh conductive layer; (d) forming a eighth conductive layer on the eighth conductive layer; (e) forming a ninth conductive layer on the ninth conductive layer; (f) forming a tenth conductive layer on the tenth conductive layer; (g) forming a eleventh conductive layer on the eleventh conductive layer;"

The first claims generated by PEGASUS receiving in input summary, summary and abstract or summary and background are the same, they show only a few syntactic differences with the ground truth.

For what concerns the first claims obtained giving to the model the abstract section or the background and abstract combination are equal to each other but present a little bit more syntactic differences with the ground, still preserving the same meaning.

For what concerns the text obtained with the background, it has strong syntactic differences from the reference text but the text is still pertinent to the patent.

Differently from all the other cases, the description input text leads to repetitive wording without an exact meaning, stuck in the same cycle of words, almost reaching the maximum number of output tokens, which is 275.

6.6.2 BigBird

In this section, the same first claim presented in the previous section is analyzed. The generated claims are ordered based on their respective input text scores. Again, in the first part, the first claims are reported, and in the second part, a brief qualitative analysis is reported.

Ground truth *"1. A bicycle wheel driving device comprising: hub axle; hub shell rotatably arranged around the hub axle, has a hub flange on an outside circumference thereof, and has a housing space inside; and motor disposed inside the hub shell and configured to rotate the hub shell with respect to the hub axle; and roller brake device at least partially arranged within the hub shell and extending outside of the hub shell."*

Summary + Abstract *"1. A bicycle wheel driving device comprising:* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the*

hub shell with respect to the hub axle; and a roller brake ***device arranged*** *within the hub shell* ***to the outside of the housing space****."*

Summary + Background *"1. A bicycle wheel driving device comprising:* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the textbfhousing space and configured to rotate the hub shell with respect to the hub axle; and a roller brake* ***device arranged*** *within the hub shell* ***to the outside of the housing space****."*

Summary *"1. A bicycle wheel driving device comprising:* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the textbfhousing space and configured to rotate the hub shell with respect to the hub axle; and a roller brake* ***device arranged*** *within the hub shell* ***to the outside of the housing space****."*

Background + Abstract *"1. A* ***motor-equipped hub*** *comprising:* ***a*** *hub axle;* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***the hub shell having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space* ***on the*** *inside* ***thereof****;* ***a*** *motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the hub shell with respect to the hub axle; and a roller brake* ***device arranged*** *within the hub shell* ***to the outside of the housing space****."*

Abstract *"1. A* ***motor-equipped hub*** *comprising:* ***a*** *hub axle;* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***the hub shell having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space* ***on the*** *inside* ***thereof****;* ***a*** *motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the hub shell with respect to the hub axle; and a roller brake* ***device arranged*** *within the hub shell* ***to the outside of the housing space****."*

Background *"1. A bicycle wheel driving device comprising: a hub having* ***a*** *hub axle* ***and a*** *hub shell* ***that covers*** *the hub axle;* ***an electric motor mounted on the hub shell; and an internal expanding drum brake device mounted on the hub shell, wherein the internal expanding drum brake device comprises: a brake drum that is fixed to the hub shell; a brake shoe that is fixed to the brake drum; and a rocking brake shoe that is fixed to the brake drum, wherein the rocking brake shoe is configured such***

that a portion of the rocking brake shoe contacts the brake drum when the internal expanding drum brake device is operated"

Description *"1.* ***A method of manufacturing a semiconductor device, comprising the steps of: providing a semiconductor substrate; forming a first electrode on the semiconductor substrate; forming a second electrode on the first electrode; forming a first insulating film on the second electrode; forming a second insulating film on the first insulating film; forming a third insulating film on the second insulating film; forming a fourth insulating film on the third insulating film; forming a fifth insulating film on the third insulating film; forming a sixth insulating film on the fourth insulating film; forming a seventh insulating film on the fifth insulating film; forming a eighth insulating film on the seventh insulating film; forming a ninth insulating film on the eighth insulating film; forming a tenth insulating film on the eighth insulating film; forming a eleventh insulating film on the ninth insulating film; forming a twelfth insulating film on the tenth insulating film; forming a eleventh insulating film on the twelfth insulating film; forming a twelfth insulating film on the eleventh insulating film; forming a twelfth insulating film on the twelfth insulating film; forming a thirteenth insulating film on the twelfth insulating film; forming a twelfth insulating film on the twelfth insulating film; forming a thirteenth insulating film on the twelfth insulating film; forming a twelfth insulating film on the twelfth insulating film.***"

The first claim generated using BigBird and feeding it with the summary, the combination of summary and abstract or the combination of summary and background leads to the same output text. Additionally, the generated texts are also equal to those generated using PEGASUS with the same input texts.

For what concers the texts generated giving to the model the combination of background and abstract or the only abstract, they are equal to each other. Also in this case the generated texts are the same generated using PEGASUS with the same inputs.

The text produced by feeding the model with the background leads to a poor prediction. The initial part is promising, but the text worsens going on, producing repetitive content.

The description section leads to the worst text generation. As in PEGASUS, it got stuck in the same cycle of words, repeating irrelevant and wrong information.

6.7 PEGASUS 5 epochs

The PEGASUS results presented so far are obtained after a single training epoch. This is done to be consistent and do a fair comparison between PEGASUS and BigBird.
Since the computational resources allow one to run jobs for a maximum of 5 days, this thesis also explores the results obtained training PEGASUS for 5 epochs.

The detailed ROUGE scores are reported in Table 6.7, whereas the BERTScore values are presented in Table 6.8.

Model input	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-Lsum
Abstract (no fp16)	63.42	47.93	56.01	59.29
Background	38.76	17.08	29.5	34.43
Description (no fp16)	31.95	15.3	25.36	28.59
Summary	71.81	60.27	66.5	68.64
Summary + Abstract	**76.7**	**65.64**	**71.73**	**73.82**
Summary + Background	75.22	63.8	70.04	72.24
Background + Abstract	57.62	40.74	49.69	53.34

Table 6.7: PEGASUS ROUGE scores obtained after 5 training epochs. Also in this case the input text leading to the best-generated text is made by the combination of summary and abstract.

Model input	BERTScore P	BERTScore R	BERTScore F1
Abstract (no fp16)	91.47	88.86	90.12
Background	86.42	83.89	85.11
Description (no fp16)	82.72	83.19	82.9
Summary	93.42	90.52	91.93
Summary + Abstract	**93.81**	**91.67**	**92.71**
Summary + Background	93.47	91.4	92.4
Background + Abstract	90.46	87.61	88.99

Table 6.8: PEGASUS BERTScore results achieved after training the model for 5 epochs. Again, the best input text is the combination of summary and abstract.

6.7.1 PEGASUS 1 epoch vs 5 epochs

Figure 6.7 shows the performances of PEGASUS after 1 and 5 training epochs. Generally, the model trained for 5 epochs slightly improves the performances obtained after a single training epoch. This is true except for the case in which the input text is the summary. In this case, indeed, a single training epoch leads to better results. Moreover, the highest performance increment depicted is obtained for the background and the lowest for the description.

Figure 6.8 depicts the performance variations obtained with the two different numbers of training epochs. Generally, the 5 training epochs are beneficial and the performance increment is of about a bit more of a point for each input text. There is a higher improvement for the background input text. The only upstream input section is the description, for this input text a single training epoch leads to slightly better results.

In general, training the PEGASUS model for 5 epochs leads to some improvement, especially for the semantic similarities with the ground truth. It is also right and proper to highlight the fact that raising the number of training epochs does not lead to certain improvements.

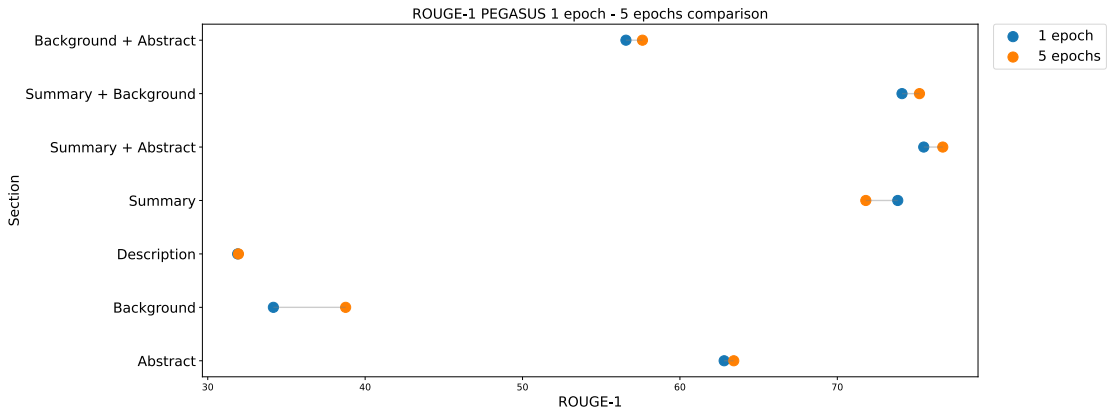


Figure 6.7: This plot depicts the ROUGE-1 performance variations due to the rise of the number of training epochs from 1 to 5 using PEGASUS. The highest improvement is obtained with the background, the lowest with the description, and a worsening with the summary.

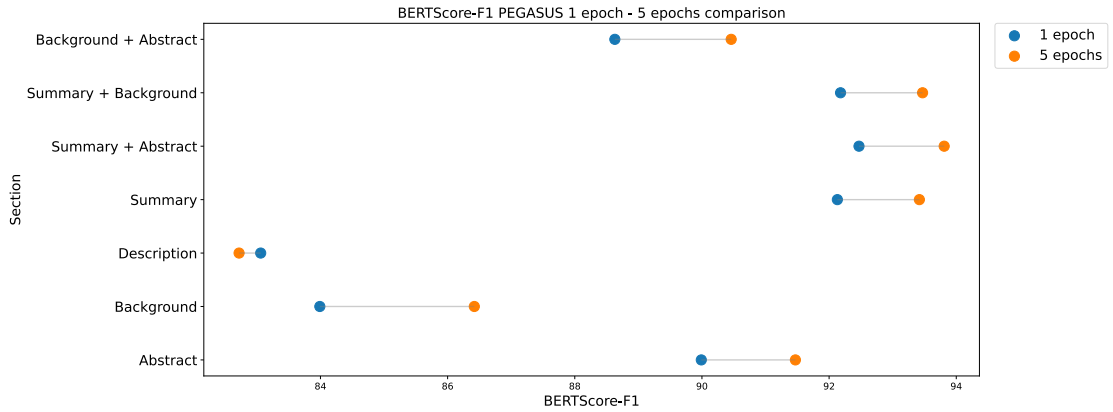


Figure 6.8: This plot shows the BERTScore F1 variation obtained training PEGASUS for 5 epochs against a single epoch. The highest improvement is obtained with the background. Instead, the performance worsens with the description input text.

6.7.2 PEGASUS 5 epochs vs BigBird 1 epoch

This section compares also the performances obtained after 5 training epochs with PEGASUS and one epoch with BigBird. This is done because they are the maximum number of epochs for which each model can be trained, according to the time limitations, and they require training times that are roughly similar. Therefore, it is appropriate to highlight that it is not a fair comparison.

Figure 6.9 displays the ROUGE-1 performances of the two models. One can notice that there is not an evident trend. More specifically, for what concerns the background section, summary section and background combined with the abstract, BigBird brings higher performances. On the opposite, the remaining four input sections achieve a higher ROUGE-1 with PEGASUS after 5 training epochs.

Figure 6.10 shows the BERTScore improvements obtained using PEGASUS after 5 training epochs against BigBird after one training epoch. The general trend is that PEGASUS improves the performances using all input texts, except for what concerns the description, with which BigBird performs better.

As a general consideration, one can notice that the more PEGASUS is trained, the more it privileges the semantic similarities. For this reason, considering that the claim generation task can be seen as an abstractive summarization task and considering that the time needed is the same used for training BigBird for a single epoch, using PEGASUS trained for 5 epochs could be the best option.

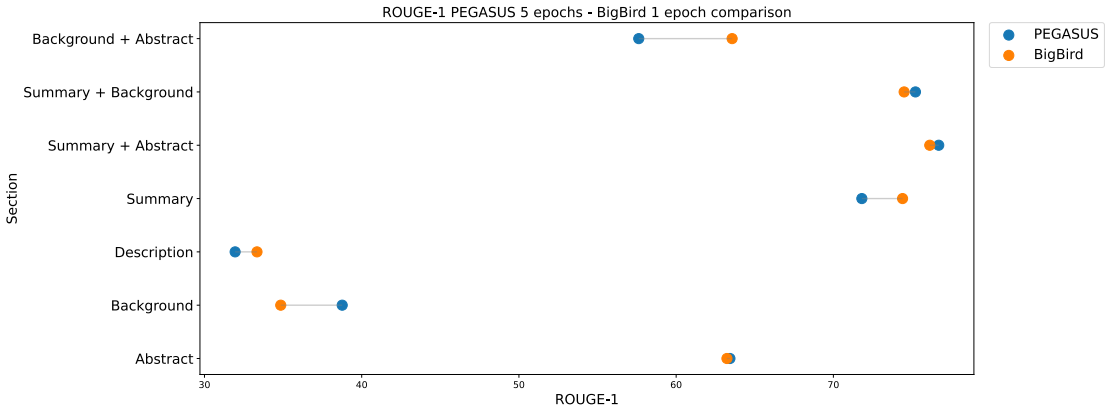


Figure 6.9: This plot depicts the ROUGE-1 performance variations obtained using PEGASUS trained for 5 epochs and those obtained with BigBird trained for a single epoch. In three out of seven cases, that are background, summary and background combined with abstract, BigBird perform better. In the remaining four cases the best model is PEGASUS.

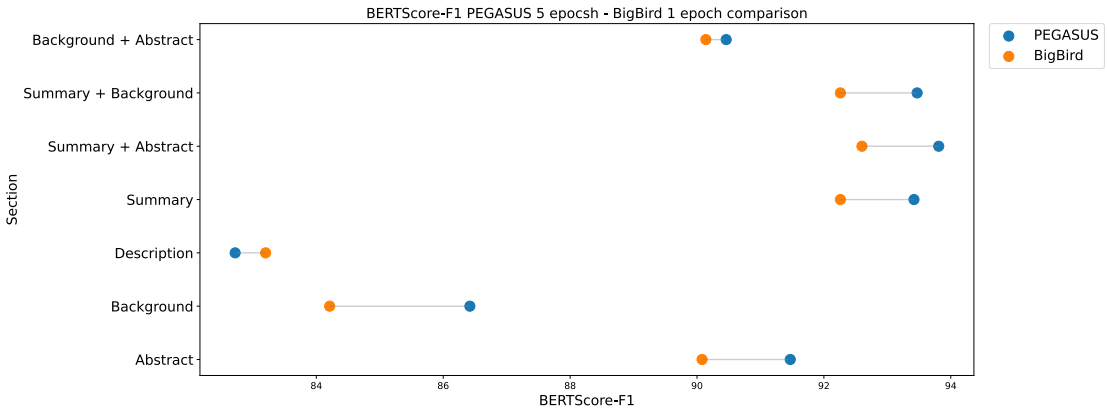


Figure 6.10: This figure displays the BERTScore-F1 improvements obtained after training PEGASUS for 5 epochs against BigBird trained for a single epoch. With all the input texts PEGASUS reaches higher scores, the only section that does not align with this trend is the description.

6.8 PRIMERA results

A last series of experiments had been carried out to determine whether performance discrepancies may be caused by a model that deals with lengthy documents but has a different architecture from PEGASUS and quantify those variations.
One has chosen to employ PRIMERA for two primary reasons: it can handle lengthy documents with up to 4,096 tokens and it uses global attention on the new special tokens. Indeed, PRIMERA is a task-specific pre-trained model, thought of as a multi-document summarization model that uses special tokens to separate the different documents. In this thesis, this model is extended and used for the summarization of single texts with several sections..
Unfortunately, there is no PRIMERA model that has been pre-trained on the patent domain. Due to this, two versions are evaluated and compared.

The first one, `allenai/PRIMERA`, is the version of PRIMERA with the only pre-training; it lacks fine-tuning. The URL for this model is `https://huggingface.co/allenai/PRIMERA` (last accessed: March 2023).

The second model used has been fine-tuned on the Multi-LexSum dataset, it is the `allenai/primera-multi_lexsum-source-long` model and is available at `https://huggingface.co/allenai/primera-multi_lexsum-source-long` (last accessed: March 2023).
Multi-LexSum is a multi-document summarizing dataset comprising summaries at three different granularities: long, short and tiny summaries. It contains documents about civil rights litigation cases [38].
As this dataset contains many legal terminologies, one picked the PRIMERA version that had been fine-tuned on it because of its resemblance to the patent domain. Nonetheless, it is right and proper to point out that the performances can be significantly impacted by the choice of fine-tuning domain. However, it is the domain with the closest fine-tuning domain among those provided by the HuggingFace hub.
Additionally, the PRIMERA version selected has been optimized for summarizing the whole source documents and producing long summaries [39].

6.8.1 Experimental setup

Unfortunately, a trained PRIMERA version is not available in the patent domain. This resulted in a remarkable decline in performance, as can be seen from Table 6.10 and Table 6.11. This demonstrates the genuine value of the pre-training stage. The reported results were obtained using mixed precision, a maximum input length of 4,096, a batch size of 16, and a learning rate of 1e-4.

Two additional experiments have been conducted to rule out the possibility that the performance decline is related to the learning rate setting or the mixed precision. First, all settings are fixed, with the exception of the learning rate, which is 1e-5. This is done in order to determine whether the 1e-4 learning rate is too high for RPIMERA. It turned out that reducing the learning rate resulted in even more performance decline.
The full precision has then been tested while maintaining the same hyperparameters, including a learning rate of 1e-4. This setting significantly increased the training time but still led to a meaningless performance increase.
The mixed precision and learning rate of 1e-4 have been used for all subsequent experiments.
The complete set of results obtained on the validation set during the hyperparameters analysis is shown in Table 6.9.

Model input	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-Lsum
Summary + Abstract (1e-4)	22.84	18.05	21.88	22.48
Summary + Abstract (no fp16)	22.89	18.14	21.94	22.53
Summary + Abstract (1e-5)	19.78	15.49	19.04	19.48

Table 6.9: Comparison of PRIMERA performance based on several hyperparameters. The results are computed using the combination of the summary and abstract as input text on the validation set. The 1e-4 and 1e-5 learning rates are compared. It is clear to see that 1e-4 produced the best outcomes. Also, the full precision has been examined. Despite the impressive rise in training time, the performance improvement may be regarded as negligible.

6.8.2 PRIMERA ROUGE and BERTScore results

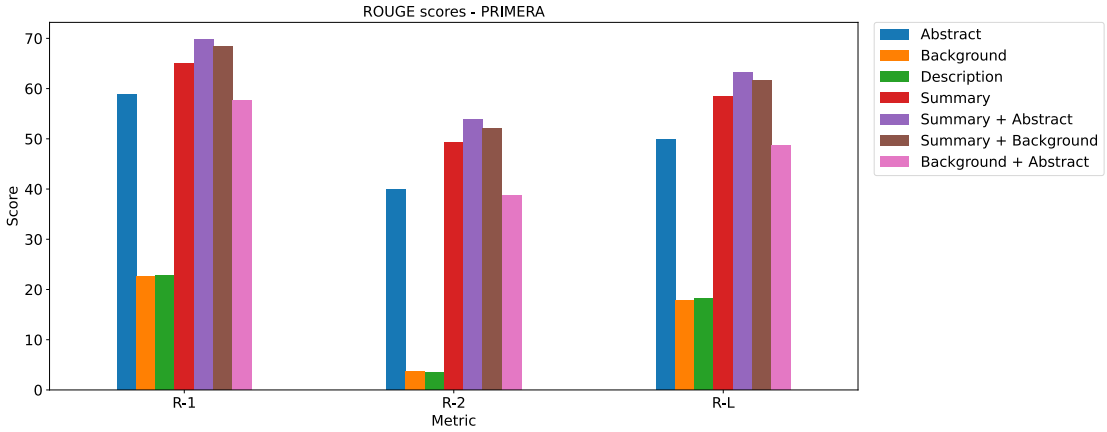
This section reports the ROUGE and BERTScore values obtained on the test set, after a single training epoch.
As it is possible to see from Table 6.10, the best ROUGE results are achieved using as input text the combination of summary and abstract. Followed by the combination of summary and background and then by the summary section. Every time the summary is included in the input text, the results are the highest, this highlights the high informativeness of the summary section.
Table 6.11 shows the BERScore results. Also for these values, the same trend can be identified.

Model input	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-Lsum
Abstract	58.75	39.93	49.94	56.0
Background	22.54	3.69	17.89	21.21
Description	22.74	3.44	18.27	20.88
Summary	65.08	49.24	58.43	62.83
Summary + Abstract	**69.7**	**53.8**	**63.12**	**67.54**
Summary + Background	68.36	52.1	61.59	66.16
Background + Abstract	57.55	38.7	48.59	54.76

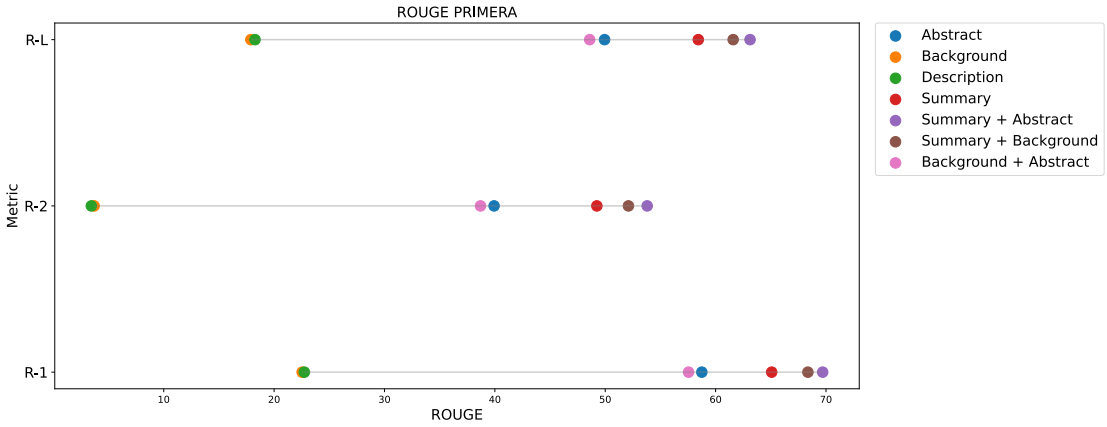
Table 6.10: PRIMERA ROUGE results achieved after training the model for 1 epoch. The best input text is the combination of summary and abstract.

Model input	BERTScore P	BERTScore R	BERTScore F1
Abstract	90.89	89.93	90.38
Background	84.23	81.87	83.02
Description	84.19	81.59	82.86
Summary	91.75	91.16	91.43
Summary + Abstract	**92.87**	**92.4**	**92.62**
Summary + Background	92.59	92.12	92.33
Background + Abstract	90.15	89.54	89.81

Table 6.11: PRIMERA BERTScore results achieved after training the model for 1 epoch. The best input text is the combination of summary and abstract.

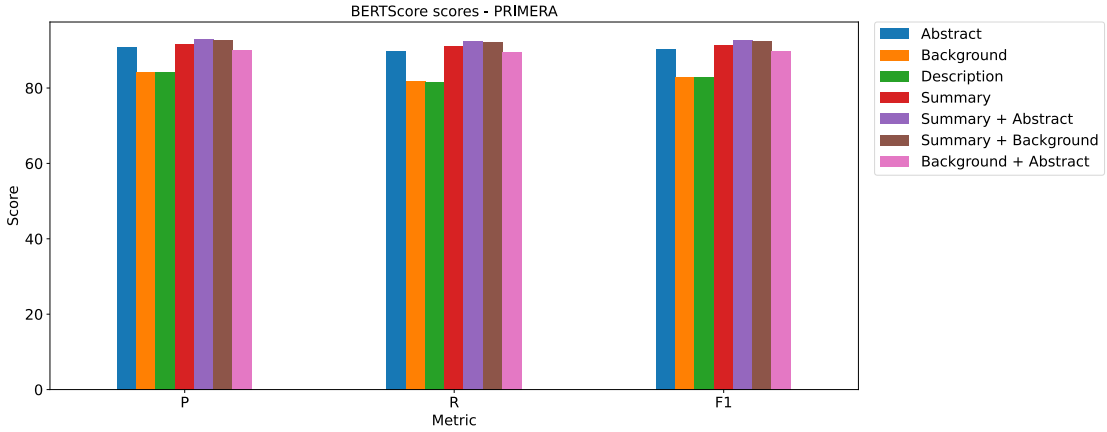


(a) Global view

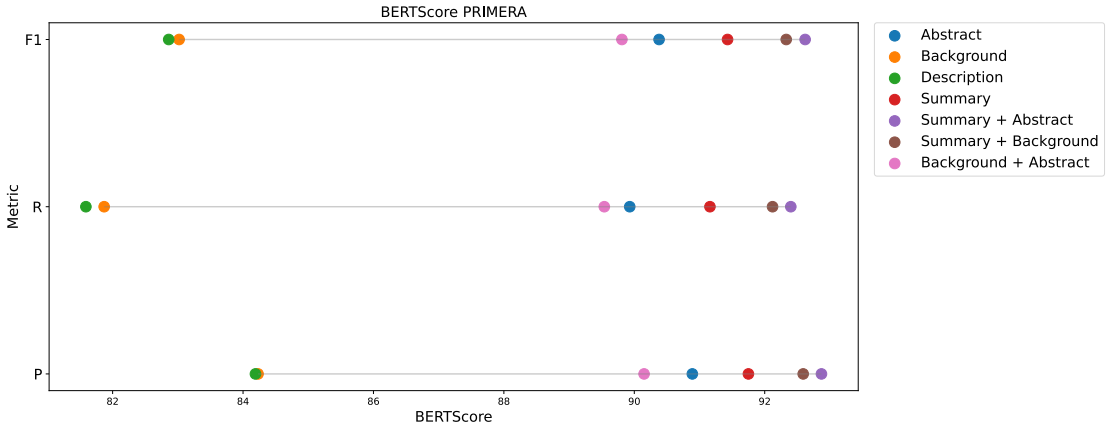


(b) Detailed view

Figure 6.11: Graphical representation of the ROUGE scores achieved by PRIMERA and reported in Table 6.10. The highest performances obtained for all the ROUGE-1, ROUGE-2 and ROUGE-L scores are reached by the summary and abstract combination, at the second place there is the combination of summary and background, followed by the summary section. The least informative input sections are the background and the description.



(a) Global view



(b) Detailed view

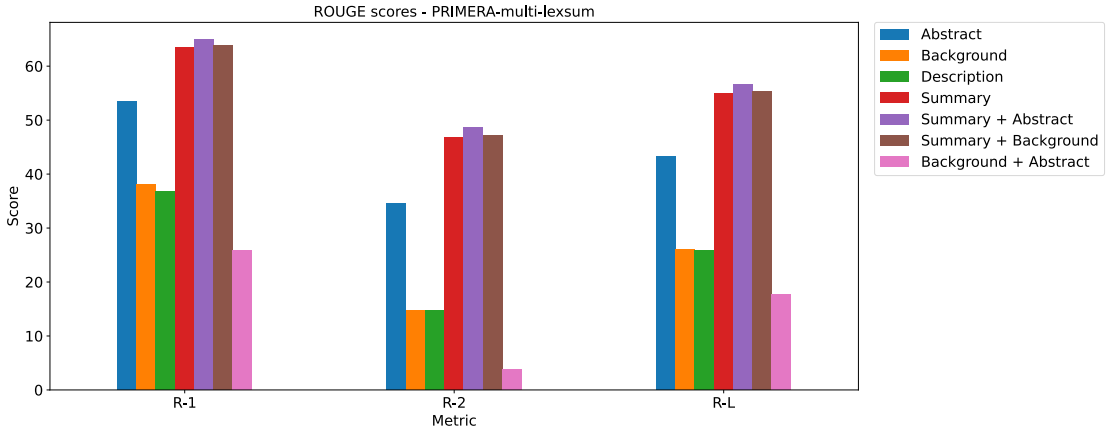
Figure 6.12: Graphical representation of the BERTScore values achieved by PRIMERA and reported in Table 6.11. Figure (a) reports the global situation of the scores, whereas Figure (b) allows us to better inspect the relative position of each input text. The best input is the summary and abstract combination, followed by a combination of summary and background and then by the summary single section input. The least informative input sections turned out to be the background and the description.

6.8.3 PRIMERA-multi-lexsum ROUGE and BERTScore results

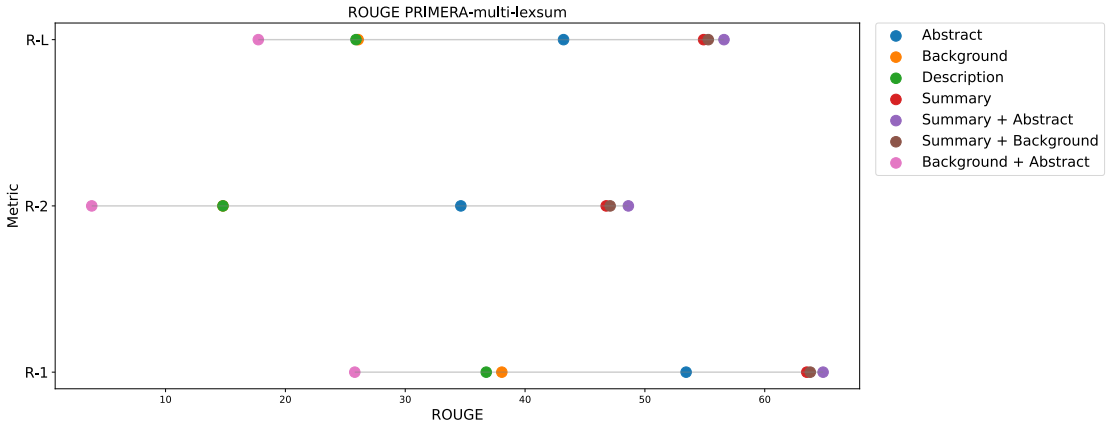
This section presents ROUGE and BERTScore results obtained using PRIMERA pre-trained on Multi-LexSum after a single training epoch on CMUmine.
Also using this model the best results are achieved by giving in input to the model the summary and abstract combination, as it is reported in Table 6.12 and 6.13. This input text is followed, in terms of performance, almost equally, by the summary plus abstract and only summary inputs. Again, each time that the summary is present in the input the performances are the highest achieved.
Using this model, differently from all the other models, the least informative section is the combination of background and abstract.
During the training, it turned out that the loss obtained with the background input text with the learning of 1e-4 explodes to NaN. For this reason, the learning rate had been decreased to 1e-5.

Model input	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-Lsum
Abstract	53.44	34.64	43.21	50.66
Background (1e-5)	38.06	14.79	26.07	34.88
Description	36.76	14.79	25.89	33.84
Summary	63.51	46.77	54.9	61.16
Summary + Abstract	**64.87**	**48.62**	**56.6**	**62.59**
Summary + Background	63.8	47.1	55.29	61.47
Background + Abstract	25.79	3.84	17.74	23.73

Table 6.12: PRIMERA-multi-lexsum ROUGE results achieved after training the model for 1 epoch. The best input text is the combination of summary and abstract.



(a) Global view

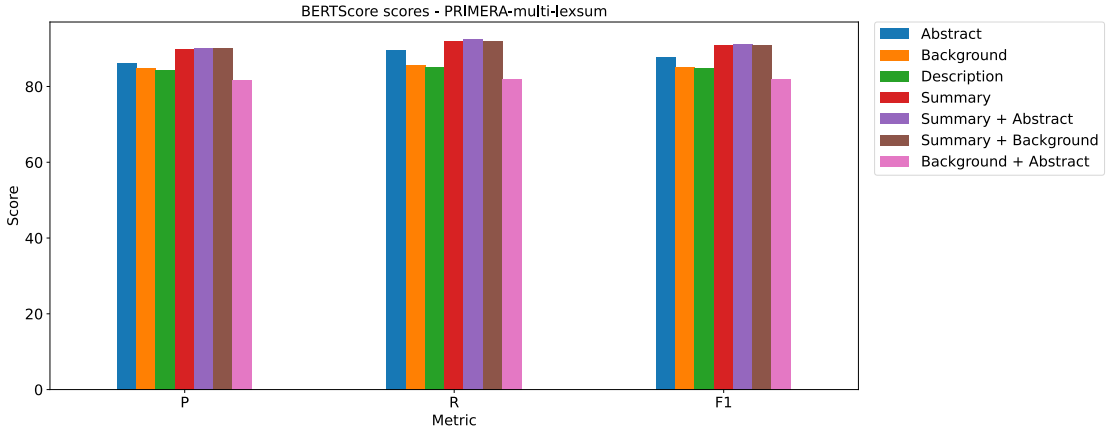


(b) Detailed view

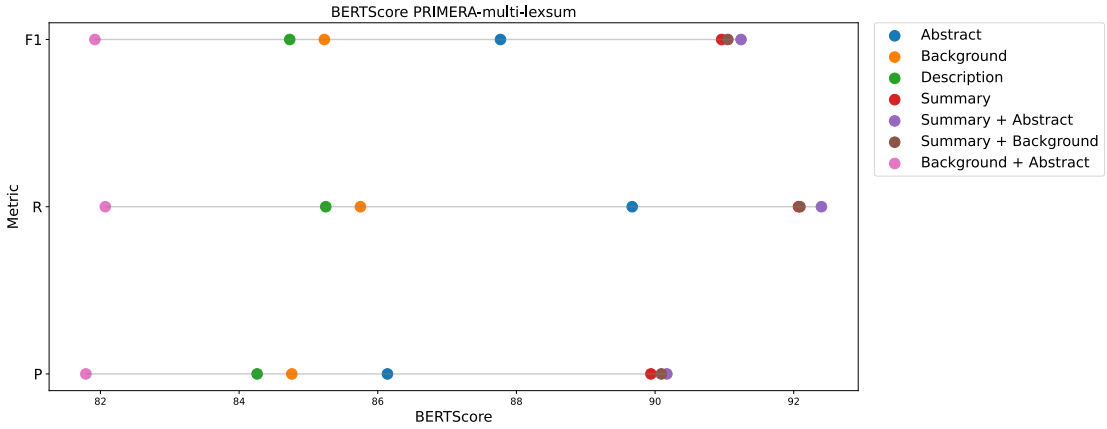
Figure 6.13: Graphical representation of the ROUGE scores achieved by PRIMERA-multi-lexsum and reported in Table 6.12. The highest performances are reached by the summary and abstract combination, followed almost equally by the combination of summary and background and the summary section. The least informative input section turned out to be the combination of background and abstract.

Model input	BERTScore P	BERTScore R	BERTScore F1
Abstract	86.14	89.67	87.77
Background (1e-5)	84.76	85.75	85.23
Description	84.26	85.25	84.73
Summary	89.94	92.07	90.96
Summary + Abstract	**90.17**	**92.4**	**91.24**
Summary + Background	90.09	92.09	91.05
Background + Abstract	81.79	82.07	81.92

Table 6.13: PRIMERA-multi-lexsum BERTScore results achieved after training the model for 1 epoch.



(a) Global view



(b) Detailed view

Figure 6.14: Graphical representation of the BERTScore values achieved by PRIMERA-multi-lexsum and reported in Table 6.13. Figure (a) reports the global situation of the scores, whereas Figure (b) allows us to better inspect the relative position of each input text. The best input is the summary and abstract combination, followed equally by the combination of summary and background and by the summary single section input. The least informative input section turned out to be the background and abstract combination.

6.8.4 PRIMERA vs PRIMERA Multi-LexSum

As it is possible to notice from Figure 6.15 there is not a clear trend between the performances of the two models. The biggest performance improvement can be identified when using the background combined with the abstract as input text and PRIMERA without finetuning. Indeed, PRIMERA-multi-lexsum achieves remarkably low scores when dealing with background plus abstract. In general, PRIMERA exceeds the scores of PRIMERA-multi-lexsum with many input texts, the only exceptions are the background and the description. For these two input texts, the performance is significantly improved by the finetuning operation.

Model input	ROUGE-1		ROUGE-2		ROUGE-L		ROUGE-Lsum	
	Multi-LexSum	PRIMERA	Multi-LexSum	PRIMERA	Multi-LexSum	PRIMERA	Multi-LexSum	PRIMERA
Abstract	53.44	58.75	34.64	39.93	43.21	49.94	50.66	56.0
Background	38.06	22.54	14.79	3.69	26.07	17.89	34.88	21.21
Description	36.76	22.74	14.79	3.44	25.89	18.27	33.84	20.88
Summary	63.51	65.08	46.77	49.24	54.9	58.43	61.16	62.83
Summary + Abstract	64.87	69.7	48.62	53.8	56.6	63.12	62.59	67.54
Summary + Background	63.8	68.36	47.1	52.1	55.29	61.59	61.47	66.16
Background + Abstract	25.79	57.55	3.84	38.7	17.74	48.59	23.73	54.76

Table 6.14: ROUGE-1 score comparison between PRIMERA and PRIMERA finetune on Multi-LexSum.

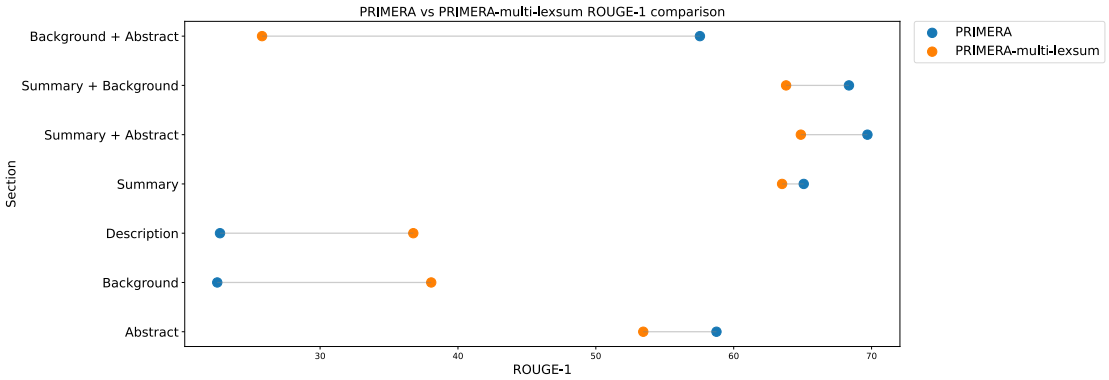


Figure 6.15: ROUGE-1 performance comparison between PRIMERA and PRIMERA-multi-lexsum.

In general, given that the finetuning domain is not a patent domain, but a legal one, it does not lead to remarkable performance improvements. On the contrary, in many cases it leads to a performance decrease, highlighting the importance of the finetuning operation on a proper domain.

6.8.5 Training times

The training times required by PRIMERA and PRIMERA-multi-lexsum to complete a single epoch with various input texts are listed in Table 6.15. PRIMERA completes a training epoch in slightly more than half the time needed by BigBird. It has the same maximum input length but allows a slightly higher batch size. In contrast to PEGASUS, it requires additional 20 hours on average.
In addition, as it is possible to notice, the fine-tuned model is a bit slower and takes about three hours more compared to PRIMERA with only pre-training.

Model input	PRIMERA	PRIMERA-lexsum
Abstract	38:08:35	41:23:05
Background	38:02:45	43:47:06
Description	38:12:22	41:19:56
Summary	37:58:13	44:00:52
Summary + Abstract	38:34:40	40:57:48
Summary + Background	38:50:27	40:56:10
Background + Abstract	38:27:00	40:16:21

Table 6.15: This table reports the training times necessary for training PRIMERA and PRIMERA-multi-lexsum for a single epoch. Both the models use a batch size of 16, a learning rate of 1e-4, mixed precision and a maximum input length of 4,096.

6.8.6 Performance difference between test and validation set

During the analysis of the results, it has been clear that the performance difference between the validation and test set obtained using PRIMERA is remarkable. This performance gap is not as evident for PRIMERA-multi-lexsum, PEGASUS and BigBird.
As it is possible to notice from the data reported in Table 6.16, for each input text the performance obtained on the validation set is remarkably lower. It is also evident from Figure 6.16 there is a smaller performance gap for the background and description input section, whereas it is almost the same for all the other input texts.
For completeness and a better understanding of the performance difference between the validation and test set for the four used models, all the ROUGE scores are reported. Table 6.18 shows the performance difference between the validation and test set for PEGASUS. Table 6.19 reports the same data regarding BigBird.

Model input	ROUGE-1		ROUGE-2		ROUGE-L		ROUGE-Lsum	
	Validation	Test	Validation	Test	Validation	Test	Validation	Test
Abstract	21.14	58.75	14.9	39.93	19.77	49.94	20.65	56.0
Background	13.39	22.54	3.1	3.69	11.32	17.89	12.99	21.21
Description	13.48	22.74	3.09	3.44	11.43	18.27	13.09	20.88
Summary	19.82	65.08	14.94	49.24	18.78	58.43	19.4	62.83
Summary + Abstract	22.84	69.7	18.05	53.8	21.88	63.12	22.48	67.54
Summary + Background	22.59	68.36	17.63	52.1	21.58	61.59	22.21	66.16
Background + Abstract	20.4	57.55	14.02	38.7	18.91	48.59	19.87	54.76

Table 6.16: PRIMERA ROUGE score gap between the values obtained on the validation and the test set. The smaller gaps can be identified for background and description.

Figure 6.16, 6.17, 6.18, 6.19 all depict the ROUGE-1 performance gap between validation and test set with the four used models.
As it is possible to notice from Figure 6.16 the ROUGE-1 differences obtained using PRIMERA are evident, differently from the results depicted in Figure 6.17 for PRIMERA-multi-lexsum, 6.18 for PEGASUS and 6.19 for BigBird.
Additionally, the two sections for which the score difference is less evident using PRIMERA are the background and the description. These two sections, according to both PRIMERA, PEGASUS and BigBird, are also the two least informative ones, characterized by the lowest results.
On the contrary, the only model that signals the combination of background and abstract as the least informative input text is PRIMERA-multi-lexsum.

Model input	ROUGE-1		ROUGE-2		ROUGE-L		ROUGE-Lsum	
	Validation	Test	Validation	Test	Validation	Test	Validation	Test
Abstract	53.34	53.44	34.44	34.64	43.03	43.21	50.58	50.66
Background	37.97	38.06	14.67	14.79	25.97	26.07	34.83	34.88
Description	36.84	36.76	1.48	14.79	25.94	25.89	33.92	33.84
Summary	63.79	63.51	46.92	46.77	55.11	54.9	61.4	61.16
Summary + Abstract	65.04	64.87	48.66	48.62	56.69	56.6	62.77	62.59
Summary + Background	63.98	63.8	47.14	47.1	55.4	55.29	61.61	61.47
Background + Abstract	25.86	25.79	3.87	3.84	17.77	17.74	23.83	23.73

Table 6.17: PRIMERA-lexsum ROUGE score gap between the values obtained on the validation and the test set. Almost no performance differences.

Model input	ROUGE-1		ROUGE-2		ROUGE-L		ROUGE-Lsum	
	Validation	Test	Validation	Test	Validation	Test	Validation	Test
Abstract	62.57	62.81	46.91	47.25	55.15	55.38	55.18	58.71
Background	34.43	34.17	14.96	14.96	26.7	26.56	26.73	30.56
Description	32.34	31.9	15.11	14.97	25.58	25.26	25.6	28.57
Summary	73.85	73.84	62.01	62.11	68.6	68.66	68.63	70.86
Summary + Abstract	75.43	75.49	63.87	64.1	63.87	70.44	70.33	72.58
Summary + Background	74.08	74.11	62.23	62.38	68.81	68.92	68.83	71.13
Background + Abstract	56.5	56.57	40.02	40.24	48.94	49.06	48.98	52.53

Table 6.18: Comparison of the PEGASUS ROUGE scores obtained on the validation and test set. No evident gap is present.

Model input	ROUGE-1		ROUGE-2		ROUGE-L		ROUGE-Lsum	
	Validation	Test	Validation	Test	Validation	Test	Validation	Test
Abstract	63.0	63.23	47.5	47.83	55.56	55.8	55.61	59.13
Background	35.1	34.85	15.42	15.4	27.19	27.04	27.21	31.15
Description	33.63	33.34	16.64	16.48	26.74	26.51	26.77	29.91
Summary	74.46	74.4	62.89	62.93	69.23	69.24	69.25	71.46
Summary + Abstract	76.1	76.13	64.84	64.95	71.0	71.06	71.02	73.22
Summary + Background	74.58	74.5	62.94	62.97	69.31	69.26	69.31	71.51
Background + Abstract	63.29	63.56	47.82	48.19	55.85	56.15	55.89	59.44

Table 6.19: BigBird ROUGE scores obtained on the validation and the test set are compared. No evident gap is present.

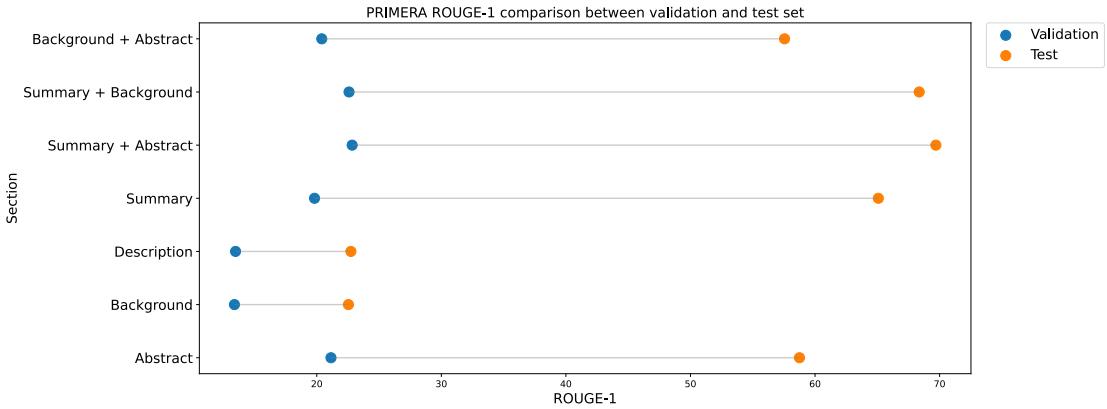


Figure 6.16: PRIMERA ROUGE-1 gap between validation and test set. The two sections that are characterized by the lower performance difference are background and description.

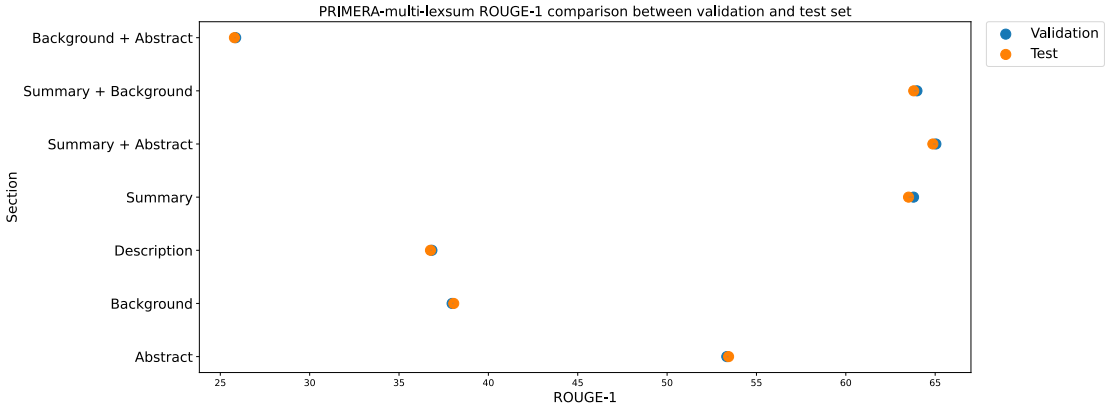


Figure 6.17: PRIMERA-multi-lexsum ROUGE-1 gap between validation and test set. There is almost no difference in performance, independently from the input text.

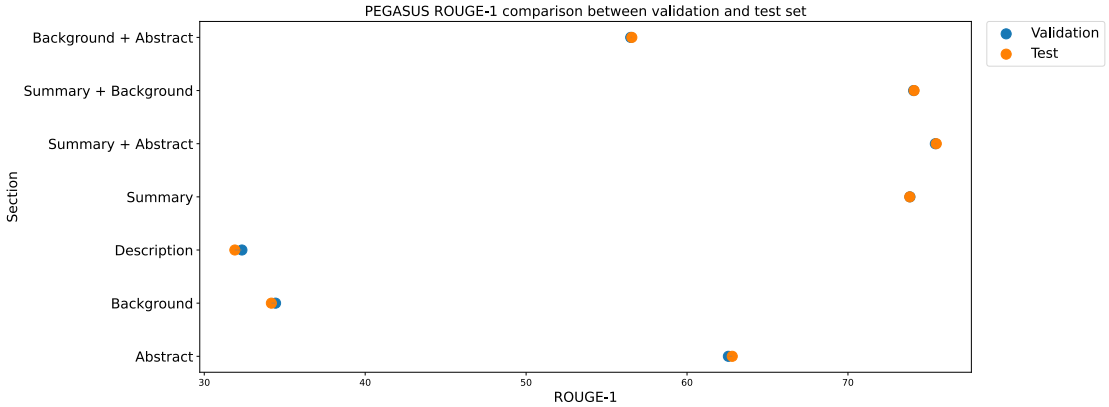


Figure 6.18: PEGASUS ROUGE-1 scores differences between validation and test set. Almost no difference for all input texts.

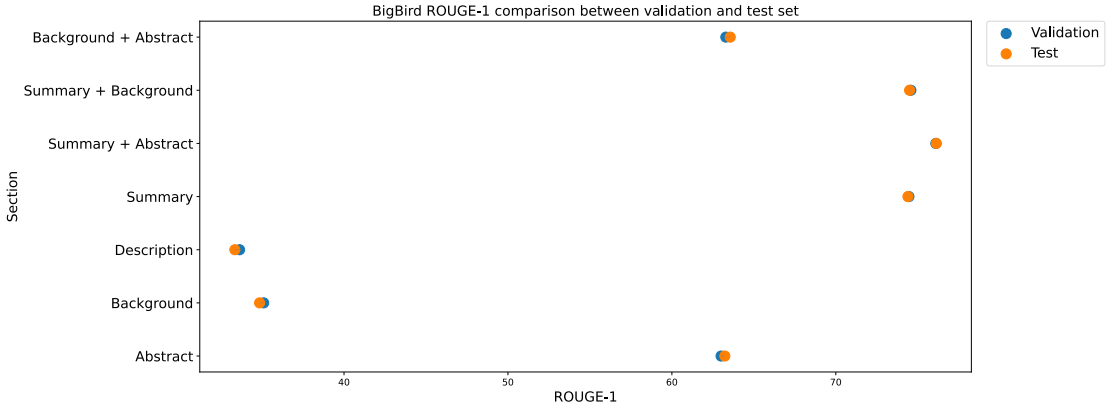


Figure 6.19: BigBird differences of ROUGE-1 scores obtains on the validation and test set. Also in this case there is almost no difference between the scores of any input section.

6.8.7 PRIMERA qualitative analysis with a textual example

In this section and in the following one, the same first claim analyzed in Section 6.6 is reported. As already stated, it is a non-cherry-picked patent. The differences that the predictions show compared to the ground truth are highlighted in bold. The results are ordered starting from the one with the highest scores, the combination of summary and abstract, to the one with the lowest scores, the description.

Ground truth *"1. A bicycle wheel driving device comprising: hub axle; hub shell rotatably arranged around the hub axle, has a hub flange on an outside circumference thereof, and has a housing space inside; and a motor disposed inside the hub shell and configured to rotate the hub shell with respect to the hub axle; and roller brake device at least partially arranged within the hub shell and extending outside of the hub shell."*

Summary + Abstract *"1. A bicycle wheel driving device comprising:* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the hub shell with respect to the hub; and a roller brake* ***device arranged*** *within the hub* ***to the outside****."*

Summary + Background *"1. A bicycle wheel driving device comprising:* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the hub shell with respect to the hub; and a roller brake* ***device arranged*** *within the hub* ***space to the outside the housing****."*

Summary *"..* *A bicycle wheel driving device comprising:****a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the hub shell with respect to the* ***axis****; and a roller brake device arranged within the hub* ***flanges to the outside circumference of the housing flange.****"*

Abstract *"1. A* ***motor-equipped hub****comprising:* ***a*** *hub axle;* ***an outer*** *shell arranged around the* ***outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space* ***on the*** *inside* ***thereof****; a motor*

drive unit housed *inside the* ***housing space*** *and configured to rotate* ***the outer*** *shell with respect to the hub axis; and* ***at least one*** *roller brake device arranged within the* ***outer sleeve of the outer casing to the outside thereof****."*

Background + Abstract ***". A wheel*** *driving device comprising:* ***a*** *hub axle;* ***an annular*** *hub shell arranged around the* ***outside of the*** *hub axle****, having a*** *hub flange on an outside circumference thereof, and* ***having*** *a housing space* ***on the*** *inside* ***thereof****; a* ***motor drive unit housed*** *inside* ***the housing space*** *and configured to rotate the hub shell with respect to the hub axis; and* ***a*** *roller brake* ***device arranged*** *within the* ***annular housing*** *shell* ***to the outside thereof****."*

Background *"1.* ***A method for providing a user interface for a computer application, the method comprising: receiving a user input; determining a user profile based on the user input and the user profile; dividing the user interface into a plurality of sections; displaying the sections; and providing a user-selectable option for each section.****"*

Dscription *"1.* ***A method for providing a user interface for a user, the method comprising: receiving a user request for a service; determining whether the user request is associated with a service provider; if the user requests are associated, providing the user interface to the user; and if it is determined that the user is not associated with the service provider, providing a service to the service user."***

6.8.8 PRIMERA Multi-LexSum qualitative analysis

Summary + Abstract *"1. A bicycle wheel driving device comprising:* ***a*** *hub axle;* ***a*** *hub shell arranged around* ***the outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the hub shell with respect to the hub* ***axles****; and a roller brake device* ***arranged within the hub shaft to the outside thereof. wherein the roller brake is configured to apply braking action against the hub drum. whereby the brake shoes are pressed by the rollers in such a fashion that the entire contact surface of each brake shoe contacts the brake drum when the brake device is operated."*

Summary + Background *"1. A bicycle wheel driving device comprising:* ***a*** *hub axle;* ***an*** *hub shell arranged around the* ***outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the hub shell with respect to the hub* ***axle****; and roller brake* ***device arranged*** *within the hub* ***flanges to the outside thereof. wherein the roller brake device comprises: a stationary bracket configured such that it can be fastened to the bicycle; a cylindrical brake drum arranged such that the brake drum can rotate integrally with the hub shaft and having an internal circumferential surface; a plurality of brake shoes that are arranged to be non-rotatable with respect the stationary bracket, each have a contact surface that can contact the braking surface, and serve to apply braking action against the brake flanges; a force applying member configured and arranged to apply forces against the brakes shoes in radially inward directions; and, brake operating unit having a plurality of rollers arranged with spaces there-between in a circumferentially oriented direction and a cam mechanism configured to press the rollers in a radially outward direction when turned, the braking operating unit being"*

Summary *"1. A bicycle wheel driving device comprising:* ***a*** *hub axle;* ***an*** *hub shell arranged around the* ***outside of*** *the hub axle,* ***having*** *a hub flange on an outside circumference thereof, and* ***having*** *a housing space inside; a motor* ***drive unit housed*** *inside the* ***housing space*** *and configured to rotate the hub shell with respect to the hub* ***axis****; and a roller brake device arranged within the hub* ***shaft to the outside the housing. wherein the hub wheel driving unit is driven by the motor drive and the bicycle is enabled to move using electric power. whereby the bicycle wheel is driven using the wheel driving*

***apparatus**."*

Abstract *".. **A motor-equipped hub** comprising: **a** hub axle; **an annular** hub shell arranged around the **outside of the** hub axle**, having a** hub flange on an outside circumference thereof, and **having** a housing space **on the** inside **thereof**; a **motor drive unit housed** inside **the housing space** and configured to rotate the **annular annular** shell with respect to the hub **axis**; and a roller brake device arranged **within the annal annular shaft shell to the outside thereof. wherein the annula annular housing space is configured to accommodate a roller brake. where R1 and R2 are each independently a hydrogen atom, a straight chain or branched alkyl group, a branching or branching alkylene group, an alkoxy group, or an alkoxyl group, and R3 and R4 are each individually a straight-chain or brankled alkylamino group, where R5 is a straight line or brannled alkenyl group or a straight straight-line or branled alicyclic group, R6 is a hydrogen atoms, a linear or brackenyl group which is a linear, branchen or cyclic group or an alicylic group which has a straight,"*

Background *"1. **A wheel** driving device comprising: **a** hub **shell having a hub** axle; **an electric motor disposed in the hub shell; a brake drum for driving a wheel of a bicycle; and brake shoes for driving the wheel of the bicycle, wherein the brake shoes are provided on the hub axle and the brake drum is fixed to the hub. the brake shoes include a first brake shoe and a second brake shoe, where R1 and R2 are independently a first and a third brake shoe; R3 and R4 are independently an elastic member; and the first brake shoes and the second brake shoes have a first contact portion and a first non-contact portion, R5 and R6 are independently the same, and the first contact portions are connected to the first brake and the third brake shoes,"*

Description *"1. A **method for providing a user interface for a computer system, comprising: receiving a request for a user-defined function from a user; determining whether the user-specified function is available for execution by the computer system; if the user is available, providing the user interface to the user; and if it is determined that the user does not have the user defined function, provides a user input to the computer, wherein the user input is selected from the group consisting of a plurality of user-specific functions, and if the plurality of the user specific functions are available, the user can select a user specific function from the plurality, wherein if the user selects a user*

selectable function, the computer provides the user with the user select interface to perform the user selected function."

Background + Abstract ***" .. A method of forming a semiconductor device, comprising: forming a first insulating film on a semiconducting substrate; patterning the first insulator film to form a first opening; forming an insulating layer on the semiconductating substrate; and performing a second insulating process to form an opening in the insulating insulating material. wherein the first opening is formed in a region of the semiconductor substrate that is not covered by the insulator layer. where R1 is a hydrogen atom or a substituent, R2 is a substitutent, and R3 is a group that is capable of reacting with a group in the semicconducting substrate."***

6.8.9 Conclusions about PRIMERA

In conclusion, it turned out that the pre-training phase is a crucial step in order to have promising results. However, it is also possible to notice that, even with a maximum input length equal to the one of BigBird, the maximum batch size supported by the computation resources is 16, against 12 of BigBird. The PRIMERA training time is significantly lower than that of BigBird.
A possible solution to deeply analyze the pre-training importance is to train from scratch PRIMERA on the patent domain. This approach requires enormous time and resources and due to the available computational resources, this analysis could not have been performed.
Another possible approach is to evaluate different fine-tuned PRIMERA versions in order to find if there is an available model fine-tuned on a domain that is close enough to the patent domain to produce some performance improvements.

Chapter 7

Conclusions and Future works

This thesis addresses the task of automating the generation of patent claims to keep up with the filing rate of patents at patent offices, such as the USPTO. The task is addressed as an abstractive summarization task, and the study focuses on identifying the most informative patent sections for claim generation and how the input length affects model performance evaluated by means of ROUGE and BERTScore.

Two transformer-based abstractive summarization models, PEGASUS and BigBird, with different attention mechanisms for analyzing different input lengths, were tested on seven input texts, including four single sections (abstract, summary, background, and description) and three combinations of two sections (summary and abstract, summary and background, and background and abstract).

In order to allow the models to be aware of the semantic content of the sections, their boundaries are highlighted by means of several special tokens, that are concatenated at the beginning and the end of each section.

The results suggest that the combination of the summary and abstract is the most informative input text, while the description is the least informative section. The best performances are achieved using BigBird, which is designed to handle longer input texts, although it requires a longer training time. The evaluation metrics used are ROUGE, which assesses syntactic similarities, and BERTScore, which prioritizes semantic similarities. Notably, the study found that the generated summaries achieved impressive ROUGE scores, despite the fact that the task involved generating new text, and ROUGE is typically used for extractive

summarization tasks. This is a significant result that highlights the effectiveness of the summarization techniques used in this thesis.

The last part of the study involves the analysis of a model that allows using global attention on the special tokens used to define the section boundaries but that is not pre-trained on the patent domain. The model in question is PRIMERA. The analysis highlights the significance of the pre-training phase for the aim of this thesis. Indeed, despite the global attention on the special tokens, the performances achieved are lower than those obtained with the other two tested models. Additionally, it has been tested a PRIMERA model fine-tuned on Multi-LexSum, a summarization dataset containing civil rights litigation cases. It is the PRIMERA fine-tuning domain that is most similar to the patent domain available on the HuggingFace hub.

Future work could involve deleting documents without descriptions, extending the work to deal with claims other than the first one, checking if changing the section order affects performance, performing hyperparameter tuning, and training the model on HUPD.
However, the most interesting perspective is to train PRIMERA from scratch on the patent domain, and then analyze if the global attention on special tokens, together with the pre-training, could allow it to surpass the ROUGE and BERTScore obtained so far. This approach requires significant time and resources, and could not have been done with the available HPC resources. In absence of the required resources, another possible solution is to try several fine-tuned models, analysing if there is a fine-tuning domain closer enough to the patent domain to allow a performance improvement.

Bibliography

[1] Wikipedia. *Patent — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Patent&oldid=1144866077. [Online; accessed March-2023]. 2023 (cit. on p. 1).

[2] USPTO. *Nonprovisional (Utility) Patent Application Filing Guide*. Accessed: March-2023. URL: https://www.uspto.gov/patents/basics/types-patent-applications/nonprovisional-utility-patent (cit. on p. 1).

[3] Ozan Tonguz, Yiwei Qin, Yimeng Gu, and Hyun Hannah Moon. «Automating Claim Construction in Patent Applications: The CMUmine Dataset». In: *Proceedings of the Natural Legal Language Processing Workshop 2021*. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 205–209. DOI: 10.18653/v1/2021.nllp-1.21. URL: https://aclanthology.org/2021.nllp-1.21 (cit. on pp. 2, 27, 44–46, 49, 51).

[4] Intellectual Property Office. *Patent Factsheets: Claims*. Accessed: March-2023. URL: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/873879/WS0057_Claims_Jan_2020.pdf (cit. on p. 2).

[5] USPTO. *Statement delivered before the United States Senate Subcommittee on Intellectual Property Committee on the Judiciary*. Accessed: March-2023. URL: https://www.uspto.gov/about-us/news-updates/statement-commissioner-patents-andrew-hirshfeld-united-states-senate (cit. on p. 2).

[6] Wikipedia. *Natural language processing — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Natural%20language%20processing&oldid=1143695557. [Online; accessed March-2023]. 2023 (cit. on p. 2).

[7] Eva Sharma, Chen Li, and Lu Wang. *BIGPATENT: A Large-Scale Dataset for Abstractive and Coherent Summarization*. 2019. DOI: 10.48550/ARXIV.1906.03741. URL: https://arxiv.org/abs/1906.03741 (cit. on pp. 3, 26, 29, 43, 44, 56).

[8] Wikipedia. *Word embedding — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Word%20embedding&oldid=1146771404. [Online; accessed March-2023]. 2023 (cit. on p. 8).

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781 (cit. on pp. 9, 10).

[10] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. *Enriching Word Vectors with Subword Information*. 2016. DOI: 10.48550/ARXIV.1607.04606. URL: https://arxiv.org/abs/1607.04606 (cit. on p. 11).

[11] Jeffrey Pennington, Richard Socher, and Christopher Manning. «GloVe: Global Vectors for Word Representation». In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162 (cit. on p. 12).

[12] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. *Deep contextualized word representations*. 2018. DOI: 10.48550/ARXIV.1802.05365. URL: https://arxiv.org/abs/1802.05365 (cit. on p. 13).

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762 (cit. on pp. 14–16, 36, 43).

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805 (cit. on pp. 16, 33, 35, 62).

[15] Alec Radford and Karthik Narasimhan. «Improving Language Understanding by Generative Pre-Training». In: 2018 (cit. on pp. 17, 33).

[16] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. *PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization*. 2019. DOI: 10.48550/ARXIV.1912.08777. URL: https://arxiv.org/abs/1912.08777 (cit. on pp. 18–20, 33, 38, 43, 44, 56, 57).

[17] Manzil Zaheer et al. «Big Bird: Transformers for Longer Sequences». In: (2020). DOI: 10.48550/ARXIV.2007.14062. URL: https://arxiv.org/abs/2007.14062 (cit. on pp. 19, 21, 33, 56).

[18] Duncan J. Watts and Steven H. Strogatz. «Collective dynamics of 'small-world'networks». In: *Nature* 393.6684 (1998), pp. 440–442 (cit. on p. 20).

[19] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. DOI: 10.48550/ARXIV.1907.11692. URL: https://arxiv.org/abs/1907.11692 (cit. on p. 20).

[20] Ramesh Nallapati, Bowen Zhou, Cicero Nogueira dos santos, Caglar Gulcehre, and Bing Xiang. *Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond*. 2016. arXiv: 1602.06023 [cs.CL] (cit. on p. 24).

[21] David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. «English gigaword». In: *Linguistic Data Consortium, Philadelphia* 4.1 (2003), p. 34 (cit. on p. 24).

[22] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. «A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents». In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)* (2018). DOI: 10.18653/v1/n18-2097. URL: http://dx.doi.org/10.18653/v1/n18-2097 (cit. on p. 24).

[23] Max Grusky, Mor Naaman, and Yoav Artzi. *Newsroom: A Dataset of 1.3 Million Summaries with Diverse Extractive Strategies*. 2020. arXiv: 1804.11283 [cs.CL] (cit. on p. 25).

[24] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. «Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization». In: *ArXiv* abs/1808.08745 (2018) (cit. on p. 25).

[25] Mirac Suzgun, Luke Melas-Kyriazi, Suproteem K. Sarkar, Scott Duke Kominers, and Stuart M. Shieber. *The Harvard USPTO Patent Dataset: A Large-Scale, Well-Structured, and Multi-Purpose Corpus of Patent Applications*. 2022. URL: https://arxiv.org/abs/2207.04043 (cit. on pp. 29, 36, 46).

[26] Wikipedia. *Automatic summarization — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Automatic%20summarization&oldid=1143448629. [Online; accessed March-2023]. 2023 (cit. on p. 31).

[27] Wen Xiao, Iz Beltagy, Giuseppe Carenini, and Arman Cohan. *PRIMERA: Pyramid-based Masked Sentence Pre-training for Multi-document Summarization*. 2021. DOI: 10.48550/ARXIV.2110.08499. URL: https://arxiv.org/abs/2110.08499 (cit. on pp. 33, 38–40).

[28] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. DOI: 10.48550/ARXIV.1910.13461. URL: https://arxiv.org/abs/1910.13461 (cit. on pp. 33, 34, 36).

[29] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2019. DOI: 10.48550/ARXIV.1910.10683. URL: https://arxiv.org/abs/1910.10683 (cit. on pp. 33–35, 43).

[30] Iz Beltagy, Matthew E. Peters, and Arman Cohan. *Longformer: The Long-Document Transformer*. 2020. DOI: 10.48550/ARXIV.2004.05150. URL: https://arxiv.org/abs/2004.05150 (cit. on pp. 33, 36, 37, 39).

[31] Chin-Yew Lin. «ROUGE: A Package for Automatic Evaluation of Summaries». In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013 (cit. on pp. 38, 61).

[32] Yen-Chun Chen and Mohit Bansal. «Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting». In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 675–686. DOI: 10.18653/v1/P18-1063. URL: https://aclanthology.org/P18-1063 (cit. on pp. 41, 42).

[33] Sandeep Subramanian, Raymond Li, Jonathan Pilault, and Christopher Pal. *On Extractive and Abstractive Neural Document Summarization with Transformer Language Models*. 2019. DOI: 10.48550/ARXIV.1909.03186. URL: https://arxiv.org/abs/1909.03186 (cit. on pp. 42, 43).

[34] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. «Language Models are Unsupervised Multitask Learners». In: 2019 (cit. on p. 43).

[35] Abigail See, Peter J. Liu, and Christopher D. Manning. *Get To The Point: Summarization with Pointer-Generator Networks*. 2017. DOI: 10.48550/ARXIV.1704.04368. URL: https://arxiv.org/abs/1704.04368 (cit. on pp. 44, 45).

[36] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. *BERTScore: Evaluating Text Generation with BERT*. 2019. DOI: 10.48550/ARXIV.1904.09675. URL: https://arxiv.org/abs/1904.09675 (cit. on p. 62).

[37] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2017. DOI: 10.48550/ARXIV.1711.05101. URL: https://arxiv.org/abs/1711.05101 (cit. on p. 65).

[38] AllenAI. *Multi-LexSum: Real-World Summaries of Civil Rights Lawsuits at Multiple Granularities*. Accessed: March-2023. URL: https://multilexsum.github.io/ (cit. on p. 86).

[39] AllenAI. *Multi-LexSum: Real-World Summaries of Civil Rights Lawsuits at Multiple Granularities*. Accessed: March-2023. URL: https://github.com/multilexsum/dataset (cit. on p. 86).