FIRST LINE OF TITLE
SECOND LINE OF TITLE

THIS IS A TEMPORARY TITLE PAGE

It will be replaced for the final print by a version

provided by the service academique.

Thèse n. 1234 2011

présenté le 12 Mars 2011

à la Faculté des Sciences de Base

laboratoire SuperScience

programme doctoral en SuperScience

École Polytechnique Fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences

par

Paolino Paperino

acceptée sur proposition du jury:

Prof Name Surname, président du jury

Prof Name Surname, directeur de thèse

Prof Name Surname, rapporteur

Prof Name Surname, rapporteur

Prof Name Surname, rapporteur

Lausanne, EPFL, 2011

Contents

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Lausanne, 12 Mars 2011 M.P. A.

i

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Key words:

List of Figures

List of Tables

1 Introduction

This thesis is about the Master's Project (MP) in industry I conducted at Magma Learning Sàrl from the 28th of September 2020 to the 26th of March 2021. The MP was supervised by Dr. Maxime Gabella, Founder and CEO of Magma Learning, and Professor Martin Jaggi, Tenure Track Assistant Professor at the Machine Learning and Optimization Laboratory (MLO) at the École polytechnique fédérale de Lausanne (EPFL).

1.1 Background

Magma Learning Sàrl is a young start-up created in 2019 in Switzerland, it has the mission to radically enhance learning thanks to artificial intelligence. It is set up as a multidisciplinary research project to understand how humans learn, how machines learn, and how they can learn from each other. The company is currently active in enhancing both corporate and education learning through an AI tutor called ARI9000[I] which comes in the form of a mobile application. It automatically generates teaching material; adapts to the interests, knowledge level and memory abilities of the user; consolidates long-term retention and gives visual feedbacks on the learning progress. ARI9000 exploits machine learning techniques, in particular Natural Language Processing (NLP), to create exercises, puzzles and topic modeling visualizations.

The MLO is active in the field of machine learning, optimization algorithms and text understanding, as well as several other application domains. The alignment of the research interests of the laboratory with those of Magma Learning Sàrl in

---

[I]https://www.magmalearning.com/ari9000
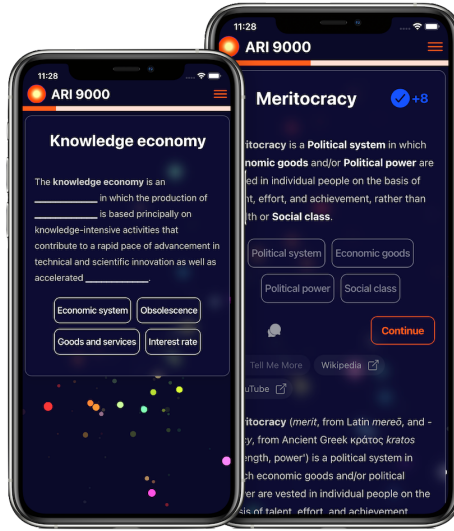
Figure 1.1: ARI9000 exercise example.

the field of NLP is the starting point for their collaboration to supervise the MP.

NLP is a discipline spanning from linguistic to computer science. It concerns the interaction between computers and human language, in particular how to automatically process, analyze and generate natural language data[II]. The result is a computer capable of understanding the lexical, syntactic and semantic levels of language. Some examples of application are summarization, automatic speech recognition, grammatical error correction, machine translation, part-of-speech tagging. The most studied tongue in NLP is English.

1.2 Problem Statement

ARI9000 produces personalized exercises thanks to a variety of algorithms. Figure 1.1 presents an exercise taken directly from the app. The starting material is either input by the user or scraped from the net and it comes in the form of natural language. A good example is a textbook. One key component in the creation of such exercises is the extraction of key passages from the material, which is often long and complex. This can be achieved with Automatic Text Summarization (ATS).

The main objective of an ATS system is to produce a summary that includes

---

[II]Natural and formal languages differ because the former is by construction explicit and non-ambiguous, while the latter is in essence implicit and ambiguous.

the main concepts in the input document in less words and to keep repetition to a minimum (Moratanch & Chitrakala, 2017; Radev et al., 2002). According to El-Kassas et al. (2020), ATS systems are designed by applying extractive, abstractive or hybrid summarization. The extractive approach selects the most important sentences from the input text and uses them to generate the summary. The abstractive approach represents the input text in an intermediate form then generates the summary with words and sentences that differ from the original text. The hybrid approach combines both extractive and abstractive approaches. ATS poses many challenges to the research and industry communities, such as identification of the most informative segments in the input text to be included in the generated summary, summarization of long documents like books, evaluation of the computer-generated summary, generation of an abstractive summary similar to a human-produced one (El-Kassas et al., 2020).

Deep learning has been successfully applied to various NLP tasks. The ATS problem is commonly solved by a Sequence-to-Sequence (Seq2Seq) model, this type of model takes a sequence of items (e.g. words, letters, tokens, ...) as input and outputs another sequence of items which makes up the summary. Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) and gated recurrent neural networks have been firmly established as state of the art approaches in sequence modeling such as ATS. A revolutionary new approach has been introduced by Vaswani et al. (2017), who proposed the Transformer, a model architecture avoiding recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output, which allows for significantly more parallelization. Transformers are pervasive and have made tremendous impact in many fields such as NLP and image processing. The attention mechanism is a key defining characteristic of Transformer models. However, a well-known concern with attention is the quadratic time and memory complexity, which can mine model scalability in many settings, such as ATS of long documents (Tay et al., 2020).

The evaluation of the computer-generated summaries is another important point in ATS. Even simple manual evaluation of summaries on a large scale over a few linguistic quality questions and content coverage as in the Document Understanding Conference (DUC) (Over, 2003) would require over 3,000 hours of human efforts (Lin, 2004). Therefore, how to evaluate summaries automatically is a key research aspect of ATS.

The MP has the objective to explore the application of ATS systems, with an ac-

cent on Transformer models, to automatically summarize and evaluate long text document in industrial environment. In particular, it tackles the main problem of Automatic Text Summarization while facing the issues of long input documents, quadratic time and space complexity of the attention mechanism and automatic evaluation of computer-generated summaries.

2 **Literature Review**

In this Chapter the theoretical concepts which make up the foundations of the MP are explained. Section 2.1 introduces the self-attention mechanism, Section 2.2 presents the most popular Transformer models in ATS and Section 2.3 focuses on the evaluation metrics in ATS.

2.1   Self-Attention and Transformer

The attention mechanism introduced by Vaswani et al. (2017) is the self-attention, also known as intra-attention. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations (Vaswani et al., 2017). The Transformer is the first model relying entirely on self-attention to compute representations of its input and output without using RNN or convolution.

In the Transformer, the encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $\boldsymbol{z} = (z_1, ..., z_n)$. Given $\boldsymbol{z}$, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next. The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 2.1, respectively.

5
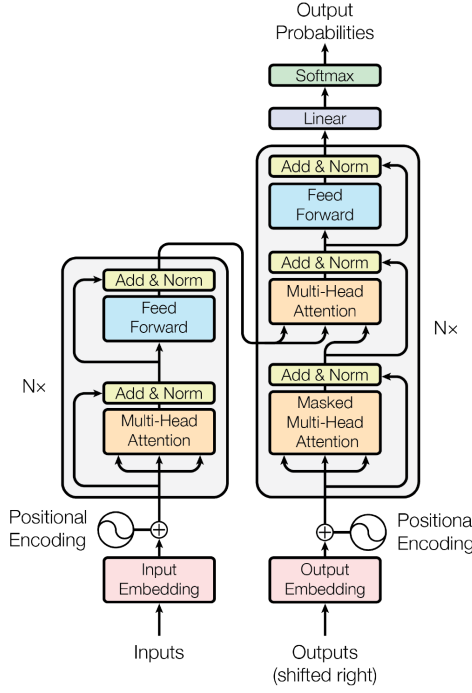
Figure 2.1: Transformer model structure.

2.1.1   Encoder and Decoder Stacks

The encoder is composed of a stack of N identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. Residual connection is employed around each of the two sub-layers, followed by layer normalization.

The decoder is also composed of a stack of N identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Residual connections around each of the sub-layers, followed by layer normalization, are used. The self-attention sub-layer in the decoder stack is masked to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i. All sub-layers in the model, as well as the embedding layers, produce outputs of dimension d_{model}, which is set to 512 by Vaswani et al. (2017).
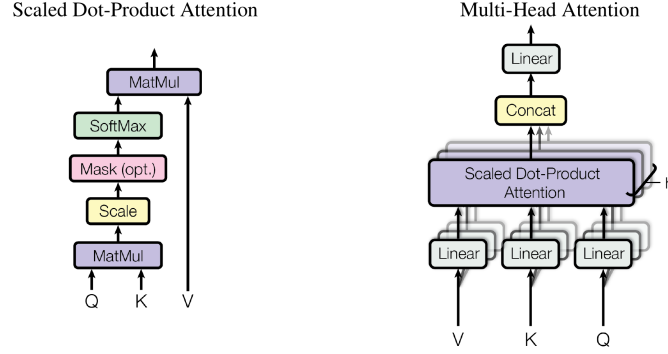
Figure 2.2: Scaled Dot-Product Attention (left) and Multi-Head Attention (right).

2.1.2 Multi-Head Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values and output are vectors. The most popular attention mechanism is the one introduced by Vaswani et al. (2017) and it is called Scaled Dot-Product Attention. The input consists of queries and keys of dimension d_k and values of dimension d_v. The weights on the values are computed by applying softmax to the dot products of the query with all keys, each divided by $\sqrt{d_k}$. In practice, the attention function is calculated on a set of queries simultaneously, stacked together into a matrix Q. The keys and values are also stacked into matrices K and V:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, Vaswani et al. (2017) found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values the attention function is performed in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in the left side of Figure 2.2. The right side of Figure 2.2 shows multi-head attention, which allows the model to jointly attend to information from different representation subspaces at different positions:

$$\begin{aligned}
\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \\
\text{where} \quad \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)
\end{aligned} \tag{2.2}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$. Vaswani et al. (2017) uses $h = 8$ parallel attention heads. For each of these, $d_k = d_v = d_{model} / h = 64$.

The Transformer uses multi-head attention in three different ways:

- In encoder-decoder attention layers, depicted in the top right orange box of Figure 2.1, the queries come from the previous decoder layer, and the keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence.

- The encoder contains self-attention layers, depicted in the left orange box of Figure 2.1. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder.

- Similarly, self-attention layers in the decoder, depicted in the bottom right orange box of Figure 2.1, allow each position in the decoder to attend to all positions in the decoder up to and including that position. Leftward information flow is prevented in the decoder to preserve the auto-regressive property. This is implemented inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections.

2.1.3 Embeddings and Positional Encodings

The Transformer uses learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model}. It also uses the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities.

Since the Transformer does not contain recurrence and convolution, in order for the model to make use of the order of the sequence, positional encoding is added to the input embeddings before the encoder and decoder stack, as seen in Figure 2.1. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. Sine and cosine functions of different frequencies are typically used:

$$
\begin{aligned}
PE_{(pos,2i)} &= sin\left(pos \,/\, 10{,}000^{2i/d_{model}}\right) \\
PE_{(pos,2i+1)} &= cos\left(pos \,/\, 10{,}000^{2i/d_{model}}\right)
\end{aligned}
\tag{2.3}
$$

Where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid.

2.1.4   Transformer From Input to Output

Following the concrete example of Figure 2.1, this Section explains the step-by-step operations of a Transformer model. The input is a text document and it has length constraints due to the self-attention complexity.

The text is tokenized with either Byte Pair Encoding (BPE) (Sennrich et al., 2015) or SentencePiece, which implements both BPE and unigram language model (Kudo, 2018). The tokens vocabulary, which is made up of words and sub-words and depends on the tokenization technique, keeps spaces and punctuation into account. The vocabulary dimension d_{voc} is in the order of 10^4 entries. The tokenized text is a list of numbers where each number corresponds to a token and the length is greater than or equal to the number of words in the input text.

Next, the tokens are embedded to vectors of dimension d_{model} and summed to the positional encodings. While Vaswani et al. (2017) used sinusoidal position signal, it has recently become more common to use relative position encodings (Raffel et al., 2019).

At this point, the encoder stack is applied to the embeddings and it outputs vectors of the same dimension d_{model}. Also the decoder needs some input to function properly. The first input given to the decoder is a special token called Begin of Sentence (BoS), which communicates to the decoder to start generating text. The BoS token is embedded and fed to the decoder. The second self-attention block in the decoder stack receives also the encoded embeddings of the input text from the encoder. The decoder stack produces the first output of dimension d_{model}. The linear transformation and softmax function are used to convert the decoder output to next-token probabilities.

During fine-tuning, these probabilities are compared to the ground-truth with the cross-entropy loss function. Then, the next inputs of the decoder are the tokens of the reference summary, one at a time. On the other hand, during generation, the most probable token is the first output of the model. This token is then concatenated to the previous tokens (only the BoS token at first) and fed back to the decoder stack. The decoder loop continues as described until the special token End of Sentence (EoS) is generated. The final output of the model is a list of tokens, starting with BoS and ending with EoS.
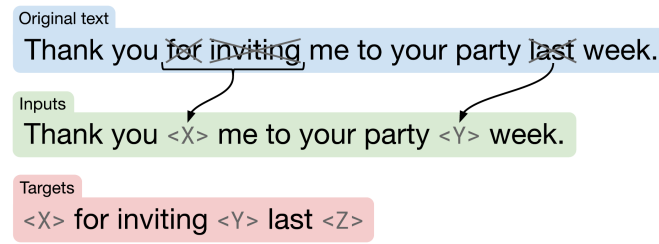
Figure 2.3: T5 unsupervised objective corrupts spans of tokens during training.

Finally, to obtain the summary, the tokens are converted into text with the inverse transformation of tokenization. The described generation technique where at each step the output is the most probable token is called greedy decoding. A more sophisticated and powerful method is beam search and it is presented in Section 3.

2.2 Transformer Models

T5 (Raffel et al., 2019), BART (Lewis et al., 2019) and PEGASUS (Zhang et al., 2020) are the three main Transformer models on which the MP focuses on. These models are the current state-of-the-art on the most common summarization datasets, such as CNN/DailyMail (CNN/DM) (Hermann et al., 2015), XSum (Narayan et al., 2018) and arXiv / PubMed (Cohan et al., 2018). T5 can perform a wide variety of English-based NLP problems, including question answering, classification and translation, to name a few. BART is focused on text generation tasks, such as question answering and summarization. PEGASUS only performs summarization and achieves state-of-the-art on 12 renowed downstream datasets.

Due to its increasing ubiquity, all of the presented models are based on the Transformer architecture introduced by Vaswani et al. (2017) and do not deviate significantly from it.

2.2.1 Training

All three Transformer models are pre-trained on a huge amount of data with an unsupervised objective. The pre-trained models are available to the public, which makes leveraging transfer learning possible. Generally, pre-training has two stages. First, the text is corrupted with an arbitrary noising function, then

a Seq2Seq model is learned to reconstruct the original text thanks to the cross-entropy loss.

T5 is pre-trained on the Colossal Clean Crawled Corpus (C4), a dataset consisting of 750 gigabytes of clean English text scraped from the web (Raffel et al., 2019). The objective specifically corrupts contiguous, randomly-spaced spans of tokens. A visual example is shown in Figure 2.3. The mean span length is 3 and the corruption rate is 15% of the original sequence. The model is pre-trained for 1 million steps on a batch size of 2^{11} sequences of length 512, corresponding to a total of about 1 trillion tokens.

BART is pre-trained on the same pre-training data as Liu et al. (2019), consisting of 160 gigabytes of news, books, stories and web text. The model is pre-trained by corrupting the input text and then optimizing the cross-entropy between the decoder's output and the original input. The corruption process masks 30% of tokens and permute all sentences. BART is pre-trained for 500,000 steps with a batch size of 8,000 ($\approx 2^{13}$).

PEGASUS is pre-trained on the weighted mixture of C4 and HugeNews, a datasets of 1.5 billion articles (3.8 terabytes) collected from news and news-like websites. The objective selects and masks whole sentences from the input documents, and concatenate the gap-sentences into a pseudo-summary. The selected sentences are the ones that appear to be important to the document. The model dynamically chooses gap sentences ratio (the number of selected gap sentences to the total number of sentences in the document) uniformly between 15% and 45%. PEGASUS is pre-trained for 1.5 million steps with a batch size of 2^{13}.

2.2.2 Low-Resource Summarization

In real-world practice, it is often difficult to collect a large number of supervised examples to train or fine-tune a summarization model. This is the case of the MP, as explained in Section 3.3. To simulate the low-resource summarization setting, Zhang et al. (2020) picked the first 10^k ($k = 1, 2, 3, 4$) training examples from each downstream dataset to fine-tune PEGASUS. In 8 out of 12 datasets, with just 100 examples, the model could be fine-tuned to generate summaries at comparable quality to a base Transformer model trained on the full supervised datasets (from 20,000 to 200,000 examples). PEGASUS also beats previous state-of-the-art results on 6 out of 12 datasets with only 1,000 fine-tuning examples.

(a) Full n^2 attention   (b) Sliding window attention   (c) Dilated sliding window   (d) Global+sliding window
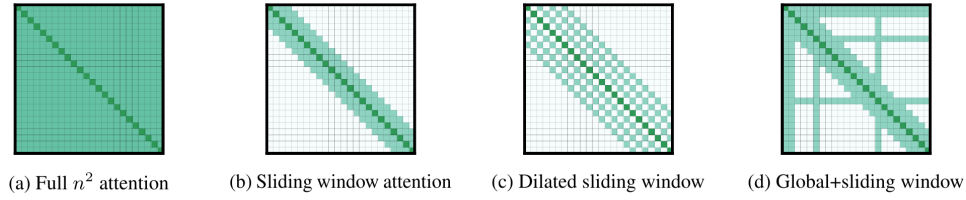
Figure 2.4: Comparing the full self-attention pattern (first image on the left) and the configuration of the attention patterns in the Longformer (from the second image owards).

2.2.3 Long Document Summarization

The Transformer success is partly due to the self-attention component which enables the network to capture contextual information from the entire sequence. While powerful, the memory and computational requirements of self-attention grow quadratically with sequence length, making it infeasible to process long sequences. Many recent models try to address this problem by introducing new attention mechanisms.

These new attention techniques can be clustered in three main groups. The data-independent patterns, employed by models such BigBird (Zaheer et al., 2020) and Longformer (Beltagy et al., 2020), make the attention matrix sparse; the data-dependent patterns, implemented in the Linformer (Wang et al., 2020) and Reformer (Kitaev et al., 2020), compress the attention matrix thanks to Locality-Sensitive Hashing (LSH), pooling and convolution; other mechanisms simplify the attention dot product with a decomposable kernel, such as Linear Transformer (Katharopoulos et al., 2020) and Performer (Choromanski et al., 2020).

Of particular interest to the MP is the Longformer Encoder Decoder (LED) (Beltagy et al., 2020), which is easily accessible online and it is initialized from BART, since both models share the exact same architecture. LED uses the Longformer sparse variant of self-attention in the encoder, presented in Figure 2.4. The decoder uses the full self-attention because the summary is by definition much shorter than the input text.

2.3 ROUGE Evaluation Metric

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) (Lin, 2004) includes measures to automatically determine the quality of a summary by com-

paring it to the ideal summary created by humans. In recent literature, ROUGE-1, ROUGE-2 and ROUGE-L F scores are unanimously used for summaries evaluation.

2.3.1 ROUGE-N

ROUGE-N, where N is typically 1 or 2, is an N-gram measure between a candidate summary and the reference summary. The ROUGE-N precision and recall, respectively p_N and r_N, are computed as follows:

$$p_N = \frac{\sum\limits_{\text{gram}_N \in S} \text{Count}_{match}\left(\text{gram}_N\right)}{\sum\limits_{\text{gram}_N \in C} \text{Count}\left(\text{gram}_N\right)}$$

$$r_N = \frac{\sum\limits_{\text{gram}_N \in S} \text{Count}_{match}\left(\text{gram}_N\right)}{\sum\limits_{\text{gram}_N \in S} \text{Count}\left(\text{gram}_N\right)}$$

(2.4)

Where N stands for the length of the N-gram gram_N, S is the reference summary, C is a candidate summary and $\text{Count}_{match}(\text{gram}_N)$ is the maximum number of N-grams co-occurring in C and S. Then, the F score is:

$$\text{ROUGE-N F score} = 2 \cdot \frac{p_N \cdot r_N}{p_N + r_N} \tag{2.5}$$

2.3.2 ROUGE-L

A sequence $Z = [z_1, z_2, ..., z_n]$ is a subsequence of another sequence $X = [x_1, x_2, ..., x_m]$ if there exists a strict increasing sequence $[i_1, i_2, ..., i_k]$ of indices of X such that for all $j = 1, 2, ..., k$, we have $x_{ij} = z_j$. Given two sequences X and Y, the Longest Common Subsequence (LCS) of X and Y is a common subsequence with maximum length.

Given a candidate summary and the reference summary, the union of the LCS is given by:

$$\text{LCS}_\cup(C, S) = \bigcup_{r_i \in S} \{w | w \in \text{LCS}(C, r_i)\} \tag{2.6}$$

Where S is the reference summary, C is a candidate summary and $LCS(C, r_i)$ is the set of LCS in C and the sentence r_i from the reference summary.

The ROUGE-L precision and recall, respectively p_L and r_L, are given by:

$$p_L = \frac{\sum\limits_{r_i \in S} |\mathrm{LCS}_\cup (C, r_i)|}{\mathrm{numWords}(S)}$$

$$r_L = \frac{\sum\limits_{r_i \in S} |\mathrm{LCS}_\cup(C, r_i)|}{\mathrm{numWords}(C)}$$

(2.7)

Where numWords($\cdot$) counts the number of words in the given text. Finally, the ROUGE-L F score between a candidate summary and the reference summary is:

$$\text{ROUGE-L F score} = 2 \cdot \frac{p_L \cdot r_L}{p_L + r_L} \tag{2.8}$$

Throughout the thesis, the terms ROUGE-1, ROUGE-2 and ROUGE-L always refer to the F score measure of the corresponding metric.

3 | Methods

In this chapter, the methods applied in the MP are explained in details. While reading the chapter, of particular interest are Appendix A, where the code written throughout the MP is presented and the functionalities of the files are described, and Appendix B, where one can find the instructions to set up a virtual environment compatible with the code. Moreover, Section 3.1 presents the most common generation technique used by Transformer models, Section 3.2 delineates the computer environment, Section 3.3 introduces the data and Section 3.4 unfolds the methodologies for using and fine-tuning the Transformer models.
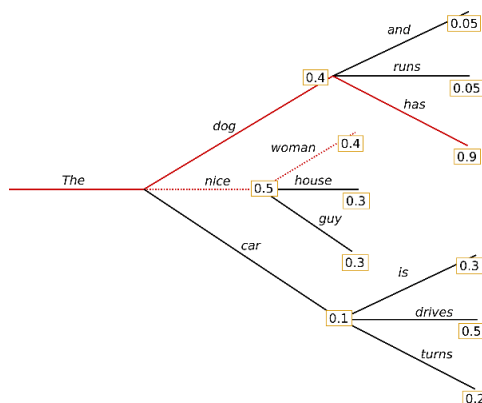
Figure 3.1: Beam search example with $num\_beams = 2$.

3.1   Beam Search

As quickly introduced in Section 2.1.4, there are several techniques for generating a summary with a Transformer. The most common one in recent literature is beam search. Beam search reduces the risk of missing hidden high probability tokens by keeping the most likely num_beams of hypotheses at each time step and eventually choosing the one that has the overall highest probability.

An example with $num\_beams = 2$ is shown in Figure 3.1. At time step 1, besides the most likely hypothesis (*The, nice*), beam search also keeps track of the second most likely one (*The, dog*). At time step 2, beam search finds that the word sequence (*The, dog, has*) with 0.36 has a higher probability than (*The, nice, woman*), which has 0.2. Beam search will always find an output sequence with higher probability than greedy search, but is not guaranteed to find the most likely output.

In addition to the num_beams parameter, other generative parameters can influence the output summary. The min_len can be used to force the model to not produce an EoS token before min_len, the max_len forces the summary length to be lower than max_len, the $length\_penalty$ is a penalty to the length of the summary, the no_repeat_ngram number forbids the generation of n-gram tokens that are already present in the input.

3.2   Environment

Python 3.6.9 is the main programming language used in the MP. Miniconda is the package manager. A virtual environment is created to facilitate reproducibility. GitHub is the version control management system. Jupyter Notebooks are used to present the code and the results. Weights and Biases is the platform employed to visualize fine-tuning progresses and results. Other important libraries that are used are Pandas for handling datasets, Gensim and Nltk for solving common NLP problems and Matplotlib and Plotly for plots. A complete list of the libraries, with the corresponding versions, can be found in the file `env.yml` of the MP repository.

3.2.1   HuggingFace

HuggingFace is an online community that helps build, train and deploy state-of-the-art models powered by the reference open source in NLP. In particular,

HuggingFace Transformers[I] is the library used in the MP to interact with the Transformer models. In more details, it is a GitHub repository that works as a container for multiple Transformer models. All the models are coded in PyTorch following a common template and they are accessible in a standardized fashion. The repository is forked and the code is personalized based on the needs of the MP.

The library is continuously updated due to bugs and new releases. To give an idea, during the period of the MP there has been an average of around 60 commits per week on the master branch, and 3,086 issues have been created. To keep pace with the dynamics of the repository and to benefit from the newly implemented features, the code has been changed multiple times. Around the end of the MP, the code is made compliant with the latest available release.

On the other hand, the community contributing to the repository is very active. The documentation is thorough and the forum is a useful communication channel, particularly thanks to the developers who participate in most of the discussions. While working at the MP, I have read more than 850 posts and written 26 posts, 12 of which were liked by the developers.

3.2.2   Cloud Computing

Fine-tuning a Transformer model requires high memory and time resources. To tackle these bottlenecks, Google Colab and Amazon Web Services (AWS) EC2 and S3 are exploited. Both are cloud computing providers. The former works with Jupyter Notebooks only, while the latter is a complete virtual machine.

To access Google Colab's GPUs, a Jupyter Notebook is simply uploaded to the service and run. The system provides one random GPU with a memory ranging from 7 to 16 gigabytes, depending on the availability.

Although AWS is more complex to set up, the GPU is chosen by the user. For the purpose of the MP the AWS EC2 g4dn.xlarge instance, with 16 gigabytes of GPU memory, and AWS S3 for data storage are chosen. In particular, the AWS machine is reachable via SSH and by accessing Jupyter Notebook remotely through a browser.

[I]github.com/huggingface/transformers

3.3   Data

The data is provided by Karger Publishers and consists of 77 books in Extensible Markup Language (XML) format, belonging to the Fast Facts series, a collection of books spanning multiple topics across the field of medicine. Each book has several chapters. At the end of each chapter there is a key facts section, consisting of a list of bullet points with the most important information of the chapter. These key fact bullet points are taken to be the summary of the chapter. The sayings "bullet point", "key fact" and "summary" refers to one single bullet point in the key facts section of a chapter. Ideally, the input of a Transformer model is a full chapter and the output is the list of bullet points of that chapter. As explained later, achieving this result is problematic.

3.3.1   Parsing and Pre-processing

The XML books are parsed using the ElementTree XML API module in Python. In total, there are 77 books. However, 8 are in Spanish, 3 in German, 3 in Italian, 1 in French, 5 do not have key facts sections and 4 are a newer edition of an already present book. Thus, 53 books are kept because they are written in English, they have one key facts section each chapter and they are the newest version available.

The XML files are parsed following the tree configuration. When a chapter root is encountered, the sub-trees containing sections and sub-sections are explored. The structure of the book is maintained during the parsing process. The key facts are retrieved using a regular expression.

At this point, some pre-processing is applied to all books. Special and Unicode characters, which can not be easily tokenized (e.g. •), are replaced. Moreover, to take care of wrongly parsed phrases, characters which should not be found at the beginning of a sentence are removed (e.g. ;,], }), and the same is done for the end of a sentence. Then, multiple spaces or new lines are merged to a single space or a single new line, respectively. Finally, words longer than 45 characters and sentences shorter than 3 words are removed.

3.3.2   Analysis

On average, there are 8.5 chapters per book and 5.6 bullet points per chapter. In total, there are 453 chapters and 2,556 bullet points. Figure 3.2a shows the distribution of the number of tokens per bullet point. The orange line shows the
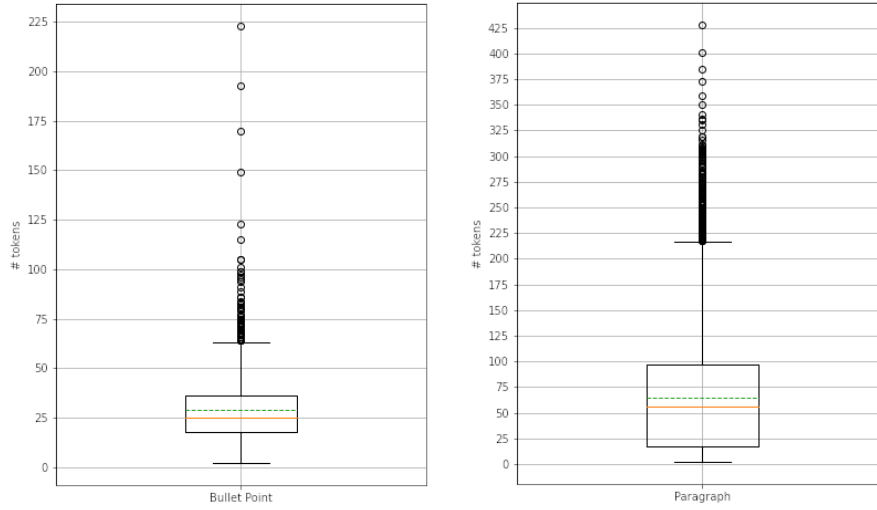
Figure 3.2: (left) Bullet points number of tokens distribution. (right) Paragraphs number of tokens distribution.

median, which is exactly 25 tokens. Although some outliers are present, these are not removed from the data since the model should learn when to generate shorter or longer summaries.

The number of tokens per chapter is 2,717 on average and has a median of 2,287. This is a problem because the maximum input length is 512 tokens for T5 and 1,024 tokens for BART and PEGASUS, due to the self-attention memory and time constraints.

The data is made up of 18,822 paragraphs in total. Figure 3.2b shows the paragraphs number of tokens distribution, which has a mean and a median of around 60 tokens. This information is useful when creating the datasets for fine-tuning.

3.3.3 Dataset Generation

Problems arise regarding the length in number of tokens of the chapters and the variability in number of bullet points per chapter. Thus, the data needs to be further processed to create a suitable dataset for the Transformer models. The only model which can handle the data out of the box is LED, Section 3.4 studies this alternative in details.

However, the other models need a specifically crafted dataset. To keep in mind

is that the goal of the MP is to automatically generate the list of bullet points of a chapter. During the MP, different ideas to create suitable datasets have been implemented and tried out. Table 3.1 lists the datasets along with an explanation of the input text and the reference summary. Hereafter, the generated datasets are presented in chronological order:

- **Chunk Chapter:** The chapter is split in chunks based on the number of tokens which can fit into the model. This number is 512 for T5 and 1,024 for BART and PEGASUS. The split is made as uniform as possible, with the constraint of never dividing in the middle of a sentence. The reference summary of each entry is the key facts section of the chapter the chunk belongs to. Using this dataset for fine-tuning is not trivial because the model is given the same key facts as reference for different input text documents. Moreover, although the model would generate one summary for each chunk, the number of chunks does not reflect the number of bullet points in the chapter. Moreover, some important information might be split and ignored in the computer-generated summary.

- **Merge or Chunk:** As in the Chunk Chapter dataset, the chapters are divided into chunks. However, in this dataset the structure of the sections is preserved as much as possible. An important drawback is that sometimes the sections are too long and must be chunked to fit into the model. As a result, this method preserves only 30% of the sections. Although the input and reference are the same as in the Chunk Chapter dataset, this time the chunks should be more meaningful. Also in this case the fine-tuning is not trivial due to the same reasons explained for the Chunk Chapter dataset.

- **Assign Bullets:** To make the fine-tuning easier, a method is implemented to assign each bullet to a specific chunk, in order to avoid the case where

Identifier	Input	Reference
Chunk Chapter	One chunk of the chapter	Key facts of the chapter
Merge or Chunk	One chunk of the chapter	Key facts of the chapter
Assign Bullets	One chunk of the chapter	Assigned key facts
Topic Modeling	Extractive summary of the chapter	Key facts of the chapter
Bullet-Paragraph	Relevant passage of the chapter	One bullet point of the chapter

Table 3.1: List of datasets with an explanation of the input and the reference.

the same reference, i.e. the key facts section of the chapter is associated with multiple chunks. In particular, a bullet point is assigned to a chunk if the ROUGE-L recall is maximal. Using this technique, around 30% of the chunks are not matched with any bullet point and they are discarded. Regarding the remaining entries, on average there are 2 bullet points per chunk, with a maximum of 11. Also in this case the fine-tuning is hard because the model would need to learn how many bullet points to output for each chunk.

- **Topic Modeling:** For the above reasons, another approach is carried out to generate a suitable dataset. For this dataset, Latent Dirichlet Allocation (LDA) and Latent Semantic Indexing (LSI) topic modeling techniques and the TextRank algorithm are employed to create an extractive summary of each chapter, called reduction. The reduction is crafted so that it can fit into the Transformer models, which then are in charge of producing the abstractive summary (i.e. the bullet points). In particular, each paragraph of the chapter is assigned an importance value based on the number of key words it contains. The final reduction is made solving a fixed-size knapsack problem, where the items are paragraphs, the weights are their number of tokens, the values are their importance and the maximum weight is set to the maximum input length of the model.

- **Bullet-Paragraph:** Fine-tuning a Transformer model to generate a single bullet point needs a dataset different from the above. When crafting the Bullet-Paragraph dataset, a single bullet point is matched with the passage in the chapter that represents it the most. Section 3.3.4 is entirely dedicated to the creation of this dataset, which is the one finally used for fine-tuning.

3.3.4 Bullet-Paragraph Dataset

The craft of this dataset comes from the need of having a cleaner dataset, with a text passage that does not exceed the input length constraint as input, and one bullet point as reference.

To create this dataset, a metric (e.g. ROUGE-L recall) is computed between every paragraph and every bullet point in the chapter. Then, each bullet point is associated to the paragraph with the maximal similarity, based on the implemented metric. As can be noted in Figure 3.2b, one paragraph, compared to a full chapter, is on average too short to generate a meaningful summary. To solve this

issue, each entry is expanded merging above or below paragraphs. This operation is done maximizing the similarity between the bullet point and the paragraph to be merged. Paragraphs are merged until the input's number of tokens is greater than or equal to 4 times the number of tokens of the matched bullet point. At this stage, each entry of the dataset is composed of the merged paragraphs as input and one bullet point as reference.

Another issue arises now because 16% of the entries have an overlap of more than 90% of tokens with another entry. Thus, one dataset (Base) is kept as is with the overlaps, and a different dataset (Merged Overlaps) is created merging paragraphs which overlap more than 90% and the corresponding bullet points. Some statistics on these two datasets is presented in Table 3.2.

To conclude, two different approaches to match paragraphs to bullet points are tried. One exploits the ROUGE-L recall and the other focuses on cosine similarity between embeddings. Three approaches are followed to create meaningful embeddings: Word2Vec, Doc2Vec and Sentence-Transformer (Reimers & Gurevych, 2019). The latter, which is based on the Transformer architecture, is the one with overall best performance. In more details, Sentence-Transformer adds a pooling operation to the output of an encoder-only Transformer to derive a fixed-size sentence embedding. The pre-trained model paraphrase-distilroberta-base-v1[II] is employed in the MP. Finally, ROUGE-L recall and cosine similarity between embeddings are compared empirically by reading 20 random bullet points and the corresponding matched passages. The first is correct 11 times out of 20 and the second 17 times out of 20. Thus, Sentence-Transformer cosine similarity is chosen as the metric to be employed for matching.

The Base and Merged Overlap datasets are then shuffled and split using train/-

---

[II]sbert.net/docs/pretrained_models.html#paraphrase-identification

	Base	**Merged Overlaps**
# of entries	2,556	2,074
Input # of tokens (mean ± var)	207 ± 94	215 ± 96
Input max # of tokens	647	647
Reference # of tokens (mean ± var)	35 ± 20	44 ± 31

Table 3.2: Statistics about the Base and the Merged Overlaps versions of the Bullet-Paragraph dataset.
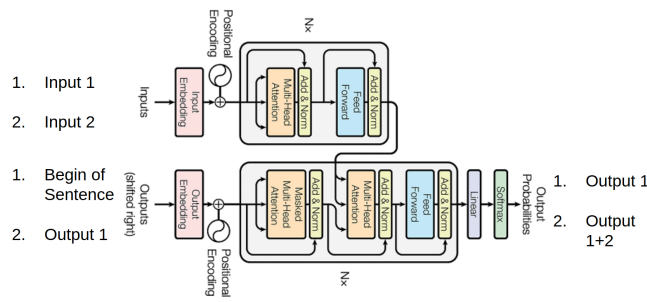
Figure 3.3: Recurrent decoder method schematized.

validation/test ratio of 80/10/10. Two different splitting techniques are used. The first ensures that bullet points belonging to the same section in the same chapter cannot belong to two different splits. The second has a stronger constraint because it ensures that bullet points belonging to the same book can not belong to two different splits. Chapter 4 presents the outcomes of choosing one technique rather than the other.

3.4   Transformer Models

Throughout the MP, different Transformer models have been tested and fine-tuned. The focus has been simultaneously on two different topics. The first, explored in the following Section 3.4.1 and 3.4.2, is about summarizing an entire chapter with the problem of having a very long input. The second is about investigating different datasets, as described in Section 3.3.3 and 3.3.4, applying and fine-tuning state-of-the-art Transformer models that generally have input constraints, as outlined hereafter in Section 3.4.3 and 3.4.4.

3.4.1   Recurrent Decoder

To overcome the long input problem, the recurrent architecture of the Transformer's decoder can be exploited. This technique, called Recurrent Decoder, is applied to the Chunk Chapter and Merge or Chunk datasets, introduced in Section 3.3.3. Since these datasets are composed of chunked chapters, a traditional model would generate a summary for one chunk at a time without information from the others. This is a problem because the information in a chapter is usually homogeneous and the generation of the bullet points should take all chunks of the chapter into account.

In the Recurrent Decoder method the first chunk C_1 is fed to the model, which generates the first summary S_1 starting from the Begin of Sentence (BoS) special token. From the second chunk onwards $\{C_2, ..., C_n\}$, although the input of the model is still the chunk only, the decoder is made to start the generation from the concatenation of the previously generated summaries. In mathematical formulas:

$$S_1 = \mathrm{decode}\,(\mathrm{encode}(C_1), \mathrm{BoS})$$
$$S_n = \mathrm{decode}\,(\mathrm{encode}(C_n), \mathrm{concat}(S1, ..., S_{n-1}))$$

(3.1)

Where BoS is the Begin of Sentence token, introduced in Section 2.1.4. A schematized visualization of the procedure is presented in Figure 3.3.

Using this method, the final summary is compared to the concatenation of the bullet points of the chapter. Although this techniques works better than an usual Transformer, it cannot reproduce the original bullet points structure. Moreover, the generative parameters, such as the min_len, must be changed every time a summary is created. This last point also makes the fine-tuning hard.

3.4.2 Longformer Encoder Decoder

The LED, introduced in Section 2.2.3, exploits sparse self-attention in the encoder to process long inputs up to around 8,000 tokens. In the MP, it is applied to the data as is, after parsing and the general pre-processing described in Section 3.3.1.

LED is the only model on the HuggingFace Transformers repository which can handle such long inputs. It was added to the platform on 13th of January and multiple bugs were fixed on 8th of February. Thus, the time to experiment with the model was limited.

Although the model is not very powerful out of the box, it reaches better performance after fine-tuning. During fine-tuning, the input of the model is a chapter and the reference is the concatenation of bullet points of the chapter with a special token <BUL> added at the beginning of each bullet point. Thanks to fine-tuning, the model should learn to output this special token in its summary, which is then interpreted as a list of bullet points. Although the results are promising, the number of bullet points generated by the model is far from the ground truth and the implementation of an appropriate evaluation metric is not straightforward. Hence, a different approach is followed.

3.4.3   T5, BART and PEGASUS

T5, BART and PEGASUS are introduced in Section 2.2. They are the Transformer models applied to the Bullet-Paragraph dataset described in Section 3.3.4. Since T5 has an input length of maximum 512 tokens, a special dataset where the paragraphs are merged up to the model's maximum length is prepared. On the other hand, BART and PEGASUS' maximum input length is 1,024 and the entry with the longest input in the dataset has 647 tokens. Due to memory problems, a distilled version (Sanh et al., 2019) of BART is used. The distilled version of the model is referred to as BART.

3.4.4   Fine-tuning

The first step towards fine-tuning the model is understanding the best environment settings. The HuggingFace Transformers library gives a diverse variety of tools for fine-tuning a model. To make the process as smooth as possible, the fine-tuning techniques for each model are copied from the respective paper. For example T5 and PEGASUS are fine-tuned using the Adafactor optimizer while BART uses AdamW, T5's learning rate is constant while BART and PEGASUS' ones decay over time. The performance outcome is very much dependent on the dataset in play when changing the batch size and learning rate parameters. Hence, a hyperparameter search is needed.

The batch size defines the number of samples that will be propagated through the network. Since higher batch size means more input text, this number cannot be raised to values higher than 2, due to GPU memory constraints on Google Colab and the AWS instance. However, most of the Transformer models are trained and fine-tuned with much higher batch sizes, as seen in Section 2.2.1. For this reason, the Gradient Accumulation Steps (GAS) parameter must be employed. Gradient accumulation is a mechanism to split the batch of samples, used for training a neural network, into several mini-batches of samples that run sequentially. The GAS parameter is the number of steps the model runs without updating its variables while accumulating (summing) gradients of those steps and then using them to compute the variable updates. For example setting batch size to 2 and GAS to 64 simulates a batch size of $2 \cdot 64 = 128$. This technique implemented on the Transformer models is highly corroborated by the HuggingFace community.

Thanks to the hyperparameter search, the best learning rate and GAS are

selected. Then, the models are fine-tuned for a number of steps aligned to the one found in the respective paper or until over-fitting the dataset. A list of the parameters used for each model is presented in Appendix C. During the process, the Weights and Biases platform is employed to monitor the system performances, such as GPU memory occupation; the training values, such as loss and learning rate's decay; and the evaluation results. The last step is to execute a second hyperparameter search on the fine-tuned model to find the best generative parameters for the beam search, described in Section 3.1. In particular, the space of generative parameters is search for the best results on the validation split.

4 Results

In this chapter, the fine-tuning results are presented. Section 4.1 shows the reproduction of BART's fine-tuning carried out originally by Lewis et al. (2019); Section 4.2 compares the fine-tuned Transformer models and Section 4.3 explores a fully automated summarization pipeline that exploits PEGASUS, which is the model with overall best performances.

All hyperparameter searches, fine-tuning projects and plots are publicly accessible on my Weights and Biases profile[I].

4.1 BART Results on XSum

To gain confidence with the HuggingFace Transformers library, in particular with the fine-tuning code, BART is fine-tuned to reproduce the results published by Lewis et al. (2019) on the XSum dataset. The XSum dataset is chosen because on average the input length of the entries is shorter than the other datasets on which BART was fine-tuned in the paper. This translates into less GPU memory consumption during fine-tuning. On the other hand, the XSum dataset counts

---

[I]wandb.ai/marcoabrate/projects

Metric	Results MP	Results Lewis et al. (2019)
ROUGE-1	44.91	45.14
ROUGE-2	21.64	22.27
ROUGE-L	36.23	37.25

Table 4.1: BART fine-tuned on the XSum dataset evaluation results in the MP and in the paper by Lewis et al. (2019).

around 200,000 entries in the training split and around 11,000 in the validation split. The large amount of data is synonym of a long fine-tuning time. In particular, the whole process takes around 30 hours.

In the paper, the model is fine-tuned for 20,000 steps while in the MP for 4,500 steps only due to time constraints. Moreover, the GAS is set to 128 since only one batch at a time can fit in the GPU; the embeddings are freezed, which means they are not updated during fine-tuning; the process is stopped and started again from the latest available checkpoint three times; and the beam search generative parameters space is not searched for the optimal combination, as opposed to the paper.

Nonetheless, on average the results of the MP are only 0.61 points lower than the results reported by Lewis et al. (2019), as one can see in Table 4.1 along with the complete evaluation outcome.

4.2 Models Comparison

After a thorough examination of the fine-tuning code, T5, BART and PEGASUS are studied on the Bullet-Paragraph dataset introduced in Section 3.3.4. To start, a hyperparameter search to find the best GAS and learning rate combination is executed for all models. The hyperparameter search carried out for PEGASUS is shown in the parallel coordinate plot of Figure 4.1. Lower GAS numbers are not present in Figure 4.1 because already discarded previously. A similar plot is analyzed for the other models. A complete list of the Transformers' parameters used during fine-tuning can be found in Appendix C.

After this first step, the models are fine-tuned on the Bullet-Paragraph dataset
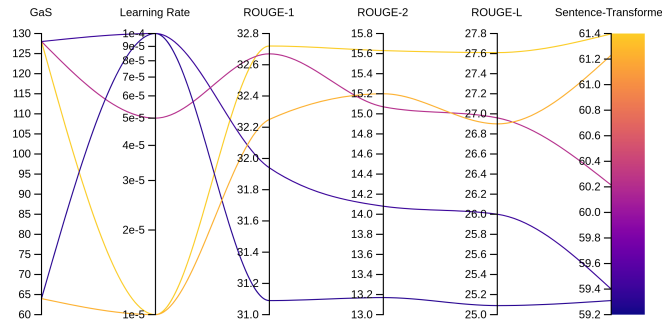
Figure 4.1: PEGASUS hyperparameter search parallel coordinate plot.
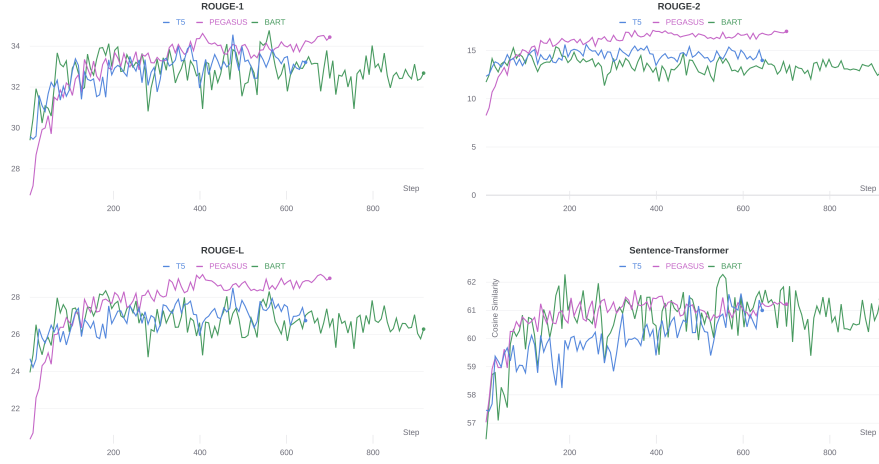
Figure 4.2: Comparison of ROUGE-1, ROUGE-2, ROUGE-L and Sentence-Transformer evaluation metrics on the validation split for T5 (in green), BART (in blue) and PEGASUS (in purple) Transformer models during fine-tuning.

using the optimal parameters. For now, the validation split refers to the one with weaker constraints. A comparison of the evaluation results of the models on the validation split during fine-tuning can be seen in Figure 4.2. The metric to the bottom right is the cosine similarity between Sentence-Transformer embeddings of the generated bullet point and the reference bullet point. This metric is included because it is also employed in the creation of the Bullet-Paragraph dataset and shows great results when comparing summaries.

It is clear from the plots of Figure 4.2 that PEGASUS, in purple, performs better than the other models. In particular, BART and T5's evaluation functions, in blue and green respectively, are noisy. This translates into the fact that choosing a checkpoint instead of another would change the quality of the summary accordingly. T5 is probably disadvantaged because of the halved input length with respect to the other models. At this point, for each model, the checkpoint achieving the best results on the validation split is chosen. Figure 4.3 shows, on the top plots and bottom left plot, the comparison of the evaluation metrics on the validation split for the models out of the box versus the best fine-tuned checkpoints. The bottom right plot of Figure 4.3 presents, in the same bar plot, the evaluation results on the validation split for T5, BART and PEGASUS' best fine-tuned checkpoints.
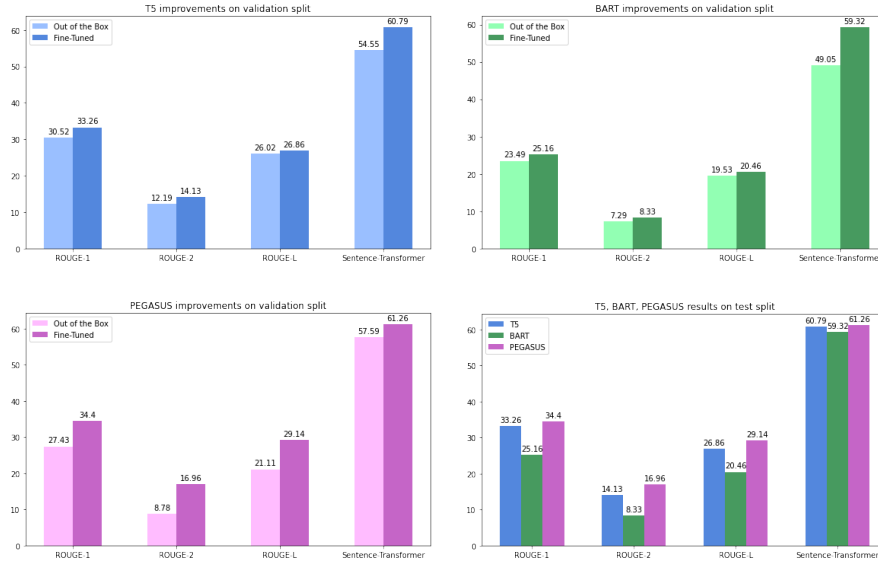
Figure 4.3: Comparison of ROUGE-1, ROUGE-2, ROUGE-L and Sentence-Transformer evaluation metrics for T5 (top left), BART (top right) and PEGA-SUS (bottom left) Transformer models out of the box (lighter colors) versus best fine-tuned checkpoints (darker colors). The bottom left plot shows the comparison of T5 (in green), BART (in blue) and PEGASUS (in purple) best fine-tuned checkpoints.

4.2.1 PEGASUS

After analyzing the fine-tuning results, PEGASUS is selected as the best alternative to generate summaries for the Bullet-Paragraph dataset. During the fine-tuning process and while choosing the best model, multiple summaries were read and compared to the input paragraphs and to the reference summary by hand. As an additional proof that the fine-tuning amazingly contributes to generate better summaries, an empirical example is selected at random from the validation split. In particular, the first entry of the split is presented hereafter. The reference bullet point is:

> The immune system has two components: innate immunity, involving mechanisms present throughout life, and adaptive (acquired) immunity, which is conferred by immune responses following exposure to an antigen, and is specific to that antigen.

PEGASUS model out of the box generates the following summary:

> A hallmark of cancer is that tumor cells - which would normally be
> recognized by the immune system as abnormal - acquire the ability
> to evade the immune system.

Which scores ROUGE-1 of 32.8, ROUGE-2 of 6.8, ROUGE-L of 16.4 and Sentence-Transformer of 52.4 compared to the reference summary. The best fine-tuned checkpoint of PEGASUS produces the following summary:

> The immune system consists of two components: innate immunity
> and adaptive immunity. Innate immunity is conferred by mecha-
> nisms that are present throughout life, such as the physical barriers
> to infection provided by the skin and mucous membranes, white
> blood cells that remove foreign material, and serum proteins such as
> lysozymes and kinins.

Which achieves ROUGE-1 of 48.8, ROUGE-2 of 23.8, ROUGE-L of 39.5 and Sentence-Transformer of 85.3 compared to the reference summary. One can notice that, although the second sentence of the fine-tuned model diverges a bit, the topic of the summary is much more aligned to the reference summary than the one generated by the model out of the box.

The last step is to execute a generative parameter search to find the best parameters for the beam search algorithm, introduced in Section 3.1. This process is not improving the results already obtained thanks to fine-tuning. However, choosing the right parameters makes it possible to adapt the generated summary to the preferences of the user. An example of how changing the $length\_penalty$ can change the evaluation results on the validation split is shown in the parallel coordinate plot of Figure 4.4. For example, one can choose to maximize the Sentence-Transformer metric while suffering a reduction of the ROUGE metrics.

To conclude, Figure 4.5 shows the fine-tuning improvements on the test split for PEGASUS.

To make the Transformer as general as possible, the fine-tuning process is repeated on the train, validation and test splits with the stronger constraints, i.e. where bullet points belonging to the same book cannot belong to two different splits. Figure 4.6 shows that PEGASUS results on the validation and test splits with stronger constraints are comparable to the results obtained on the splits with weaker constraints. Although the results are similar, the model trained on the splits with stronger constraints should be preferred because it will generalize better for unseen books.
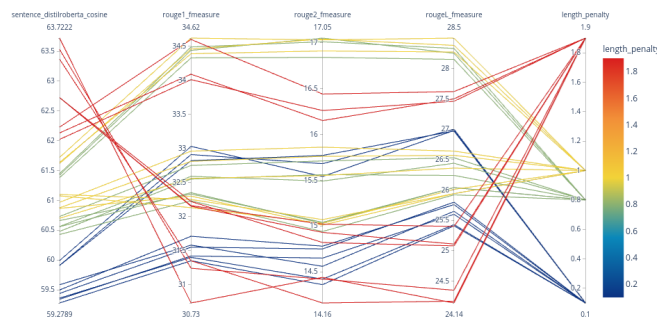
Figure 4.4: PEGASUS generative parameter search parallel coordinate plot for the *length_penalty* parameter.
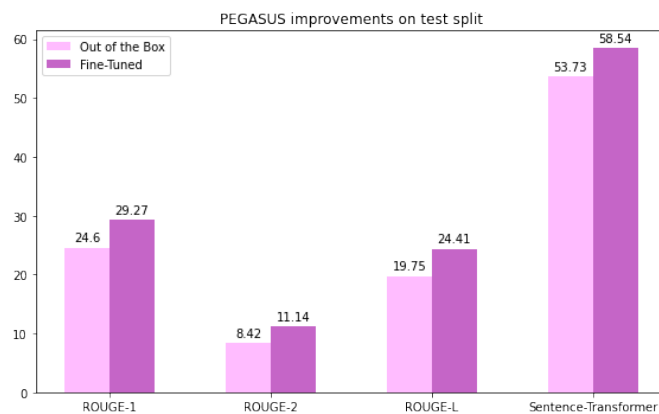
Figure 4.5: PEGASUS fine-tuning improvements on the test split.

A final experiment tests the Merged Overlaps version of the Bullet-Paragraph dataset, introduced in Section 3.3.4. This dataset is created by merging paragraphs which overlap for more than 90% of tokens and the corresponding bullet points. Figure 4.7 shows in orange the evaluation results on this dataset, compared to the results previously obtained in blue. One can clearly see that PEGASUS achieves better results on the Merged Overlaps version. This is probably due to the fact that redundant entries are not present in the new version of the dataset.

Random summarization examples picked from the test split of the Merged Overlaps Bullet-Paragraph dataset and generated by the best fine-tuned checkpoint of PEGASUS along with the reference summary and the evaluation metrics can be read in Appendix D.
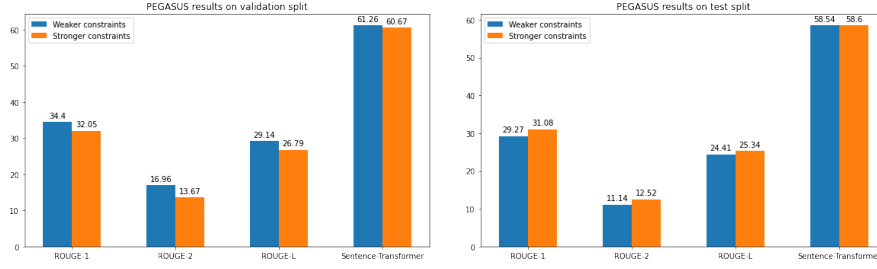
Figure 4.6: PEGASUS results on the validation (left) and test (right) splits with stronger constraints (orange) are comparable to the results obtained on the splits with weaker constraints (blue).
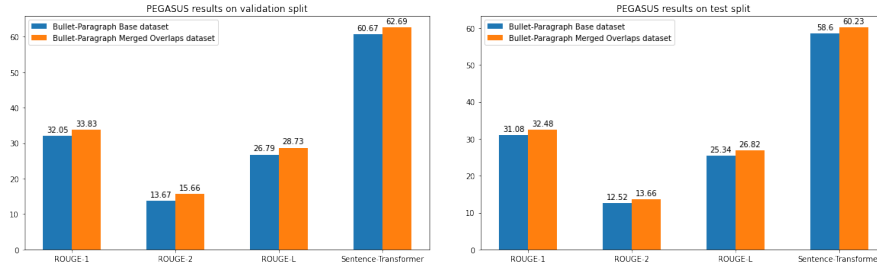
Figure 4.7: PEGASUS results on the validation (left) and test (right) splits with Bullet-Paragraph Merged Overlaps dataset (orange) and Bullet-Paragraph Base dataset (blue).

4.3 Summarization Pipeline

This section explores a fully automated summarization pipeline which can be used on new materials, such as textbooks. The pipeline heavily relies on the Sentence-Transformer metric, previously described to extract the most meaningful passages of a chapter, and on the fine-tuned PEGASUS model to create the bullet points.

To simulate a real setting, only the books belonging to the validation and test splits of the Bullet-Paragraph Merged Overlaps dataset are considered. This is done because PEGASUS is fine-tuned on the train split and otherwise the results would be biased. In this setting, the bullet points section is not employed in any way. Thus, the pipeline is an unsupervised method which can theoretically be applied to any book. Hereafter, the full process is described for a single chapter because the technique is identical for the others.

To start, one parsed and pre-processed chapter C of a Karger Publishers book belonging either to the test or validation split is retrieved. Then, the number of sections S of chapter C is considered as the most representative figure for the number of bullet points in the key facts section of the chapter. At this point, each paragraph is embedded with the Sentence-Transformer embeddings technique. In the new space, the K-Means clustering method is applied, where K equals the number of sections S of chapter C. If the chapter does not contain sections or it is impossible to parse them, the elbow technique might be employed to find the most suitable K.

Now, chapter C has S centroids. Cosine similarity is computed between paragraphs and centroids. For each centroid, the closest paragraph is extracted and selected as key paragraph. Thus, S key paragraphs are extracted from chapter C. Although these key paragraphs might be considered as the generated bullet points, PEGASUS abstractive potential is not used yet.

At this point, the key paragraphs are merged with above or below paragraphs in the text to create key passages. This process is done maximizing the cosine similarity to the associated centroid until the number of tokens of the passage is close to the maximum number of tokens that can be fed into PEGASUS. Afterwards, chapter C is associated to S key passages, which are generally made up of multiple paragraphs.

Similarly to the Merged Overlaps version of the Bullet-Paragraph dataset, passages which overlap for more than 90% of the tokens are merged. Finally, PEGASUS is applied to generate a bullet point for each key passage. As one can imagine, the number of generated bullet points with this unsupervised method diverges from the true number of bullet points. In more details, the Mean Absolute Error over the chapters in the test and validation splits is 1.7, with a variance of 1.4. However, an evaluation of this technique is carried out by concatenating both reference and prediction bullet points to create a reference and prediction summary, respectively. Table 4.2 shows ROUGE and Sentence-Transformer metrics calculated between the unsupervised prediction and reference summaries for the test and validation splits. The results, although obtained on different data, are comparable to PEGASUS fine-tuned results on the Bullet-Paragraph dataset, which means the unsupervised summaries are of similar quality.

ROUGE-1	ROUGE-2	ROUGE-L	Sentence-Transformer
35.4	12.1	32.3	67.2

Table 4.2: Unsupervised prediction summaries evaluation results on test and validation books.

5 Conclusion

In the MP, the application of Automatic Text Summarization Transformer models was explored in depth. First, the HuggingFace Transformers library was tested by reproducing BART fine-tuning results on the XSum dataset. Then, the Recurrent Decoder method was tried and the LED efficient Transformer was fine-tuned on the full data. At this point, considering the poor results, a more traditional approach was taken by applying and fine-tuning T5, BART and PEGASUS on a specifically crafted dataset. The effectiveness of fine-tuning a Transformer model was corroborated by the ROUGE evaluation metrics and the Sentence-Transformer cosine similarity between computer-generated bullet points and reference bullets points. An empirical evaluation was also carried out by me to finally choose the model with overall best performances. Then, the best fine-tuned checkpoint of PEGASUS was used to summarize the entries of the test split. Finally, a summarization pipeline was experimented to generate a list of bullet points for an unseen generic chapter, in a fully unsupervised manner.

5.1   Future Work

The new PEGASUS model fine-tuned on the Karger Publishers books will likely perform similarly with unseen Karger Publishers books or any medicine-related book. However, the model is not fine-tuned nor tested on other unrelated material. Thus, it is probable that it will perform poorly in other domains. To avoid this situation, one might add entries from different fields to the fine-tuning dataset.

The Bullet-Paragraph dataset starts from the assumption that the Sentence-Transformer cosine similarity used for matching passages to bullet points does

36

not fail. However, there is a small possibility that the wrong passage is considered and this would negatively affect the fine-tuning process. To circumvent this problem, one might look more into efficient Transformer models. These models, not yet present on the HuggingFace Transformers library except for the Longformer Encoder Decoder, are capable of taking an entire chapter as input, without the need of manually matching bullet points to shorter passages. The ones that stand out in my opinion are BigBird[I], Performer[II] and Linformer[III], which are soon to be added to the library, as you can browse from the footnotes.

Regarding the metrics, the understanding of how precise and good these metrics are when comparing summaries is still a big question for the entire community. Although the use of Sentence-Transformer cosine similarity is a little step towards a more reliable metric, this method makes absolutely impossible to understand how the metric works since the embeddings are created by a neural network with millions of parameters. The implementation of a better metric is an intricate discussion and it is outside the scope of this thesis. On the other hand, to assess how close the fine-tuned model is to human performance, a human evaluation can be conducted. For example, the Amazon Mechanical Turk service might be exploited to compare model summaries with reference summaries by asking workers to rate the summaries on a scale, as done by Zhang et al. (2020).

To conclude, if I had continued to work at Magma Learning, I would have focused more on making the code faster, lighter and easier to use for the summarization pipeline. In my humble opinion, although the Master's Project puts a solid background for Automatic Text Summarization with Transformers in an industrial environment, the process lacks velocity and consumes too much GPU memory, which results in a negative experience for the user and an expensive setting for the company, respectively.

[I]github.com/huggingface/transformers/pull/10183
[II]github.com/huggingface/transformers/pull/9325
[III]github.com/huggingface/transformers/pull/10587

Bibliography

Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. (2020). Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

Cohan, A., Dernoncourt, F., Kim, D. S., Bui, T., Kim, S., Chang, W., & Goharian, N. (2018). A discourse-aware attention model for abstractive summarization of long documents. *arXiv preprint arXiv:1804.05685*.

El-Kassas, W. S., Salama, C. R., Rafea, A. A., & Mohamed, H. K. (2020). Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, 113679.

Hermann, K. M., Kočiský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., & Blunsom, P. (2015). Teaching machines to read and comprehend. *arXiv preprint arXiv:1506.03340*.

Katharopoulos, A., Vyas, A., Pappas, N., & Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. *International Conference on Machine Learning*, 5156–5165.

Kitaev, N., Kaiser, Ł., & Levskaya, A. (2020). Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959*.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. *Text summarization branches out*, 74–81.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized bert pre-training approach. *arXiv preprint arXiv:1907.11692*.

Moratanch, N., & Chitrakala, S. (2017). A survey on extractive text summarization. *2017 international conference on computer, communication and signal processing (ICCCSP)*, 1–6.

Narayan, S., Cohen, S. B., & Lapata, M. (2018). Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*.

Over, P. (2003). An introduction to duc 2003: Intrinsic evaluation of generic news text summarization systems. *Proceedings of Document Understanding Conference 2003*.

Radev, D. R., Hovy, E., & McKeown, K. (2002). Introduction to the special issue on summarization. *Computational linguistics*, *28*(4), 399–408.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Sennrich, R., Haddow, B., & Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Tay, Y., Dehghani, M., Bahri, D., & Metzler, D. (2020). Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Wang, S., Li, B., Khabsa, M., Fang, H., & Ma, H. (2020). Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Zaheer, M., Guruganesh, G., Dubey, A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. (2020). Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*.

Zhang, J., Zhao, Y., Saleh, M., & Liu, P. (2020). Pegasus: Pre-training with extracted
    gap-sentences for abstractive summarization. *International Conference
    on Machine Learning,* 11328–11339.

A Code Structure

```
📁 magma
└📁 corpora
 └📁 Karger Fast Facts books
  └📁 77 xml
       The data, contains 77 books in XML format.
  └📁 108 pdf files
       Contains 108 PDF books, they are not considered in the MP.
  └📄 108_pdf_files.xlsx
       Table containing information about the PDF books.
└📁 datasets
 └📁 bullet_paragraph_embeddings
    Contains the Bullet-Paragraph datasets created with the embeddings.
  └📁 pegasus
   └📁 st
      Bullet-Paragraph dataset created with Sentence-Transformer embeddings.
    └📁 base
       Bullet-Paragraph dataset Base.
     └📄 df.csv
     └📄 test.csv
     └📄 test.index
     └📄 test.source
     └📄 test.target
     └📄 train.csv
     └📄 train.index
```

42

```
            │   │   ├─ 📄 train.source
            │   │   ├─ 📄 train.target
            │   │   ├─ 📄 val.csv
            │   │   ├─ 📄 val.index
            │   │   ├─ 📄 val.source
            │   │   └─ 📄 val.target
            │   ├─ 📁 merged_overlaps
            │   │
            │   │     Bullet-Paragraph dataset Merged Overlaps.
            │   │
            │   └─ 📁 df_base_selected_para
            │
            │         Dataframe with the paragraphs selected by the method.
            │
            ├─ 📁 bullet_paragraph_rouge
            │
            │   Contains the Bullet-Paragraph datasets created with ROUGE.
            │
            ├─ 📁 chunk_chapter
            │
            │   Contains the Chunk Chapter dataset.
            │
            ├─ 📁 karger_books_base
            │
            │   Contains the Karger Books data as is.
            │
            ├─ 📁 merge_or_chunk
            │
            │   Contains the Merge or Chunk dataset.
            │
            ├─ 📁 topic_modeling
            │
            │   Contains the Topic Modeling dataset.
            │
            ├─ 📄 assign_bullets.ipynb
            │
            │   Creates the Assign Bullets version of Chunk Chapter and Merge or Chunk datasets.
            │
            ├─ 📄 base_dataset.ipynb
            │
            │   Creates the Karger Books data from the XML files.
            │
            ├─ 📄 bullet_paragraph_embeddings.ipynb
            │
            │   Creates the Bullet-Paragraph datasets with embeddings.
            │
            ├─ 📄 bullet_paragraph_rouge.ipynb
            │
            │   Creates the Bullet-Paragraph datasets with ROUGE.
            │
            ├─ 📄 chunk_chapter.ipynb
            │
            │   Creates the Chunk Chapter dataset.
            │
            ├─ 📄 merge_or_chunk.ipynb
            │
            │   Creates the Merge or Chunk dataset.
            │
            ├─ 📄 topic_modeling.ipynb
            │
            │   Creates the Topic Modeling dataset.
            │
            ├─ 📁 experiments
```

```
├─ 📄 bugs.ipynb
│       Bug report file.
├─ 📄 exploration_karger_books.ipynb
│       Initial analysis of the data.
├─ 📄 generate_workflow.ipynb
│       HuggingFace Transformer summary generation in details.
├─ 📄 generative_params.ipynb
│       Explores the generative parameters for summary generation.
├─ 📄 model_config.ipynb
│       Reports the models' configurations.
├─ 📁 fine-tuning
│   ├─ 📁 config
│   ├─ 📄 ft_bart_bull_para_embed.ipynb
│   │       Fine-tunes BART on Bullet-Paragraph dataset.
│   ├─ 📄 ft_bart_xsum.ipynb
│   │       Fine-tunes BART on XSum dataset.
│   ├─ 📄 ft_led_karger_books.ipynb
│   │       Fine-tunes LED on Karger books data as is.
│   ├─ 📄 ft_pegasus_bull_para_embed*.ipynb
│   │       Fine-tunes PEGASUS on Bullet-Paragraph datasets.
│   ├─ 📄 ft_t5_bull_para_embed.ipynb
│   │       Fine-tunes T5 on Bullet-Paragraph dataset.
│   ├─ 📄 gps_pegasus_bull_para_embed*.ipynb
│           Generative parameters search for PEGASUS on Bullet-Paragraph datasets.
├─ 📁 pipeline
│   ├─ 📄 paragraph_extraction_embeddings.ipynb
│   │       Unsupervised key paragraphs extraction with embeddings.
│   ├─ 📄 paragraph_extraction_topic_modeling.ipynb
│           Unsupervised key paragraphs extraction with topic modeling.
├─ 📁 summarization
├─ 📄 config.py
│       General configuration file.
├─ 📄 env.yml
        List of libraries with version.
```

```
├─ 📄 spec-file.txt
```
   File for Conda environment creation.
```
├─ 📄 special_char_file.txt
```
   Special and Unicode characters to be replaced.
```
├─ 📄 tokenizers.pdf
```
   A list of the tokenizers used by the models.
```
├─ 📄 transformers_concepts.pdf
```
   Some concepts extracted from the most popular Transformers papers.

B Virtual Environment

This appendix describes the creation of a new virtual environment in which every file of the MP can be run without errors. The process outlined is done on a 64-bit Linux machine. The operating system in use is Debian 10. The process has not been tested on other machines.

First, a virtual environment must be created. For that purpose, Miniconda is used. When Miniconda is installed, clone the GitHub repository of the MP:

```
1  git clone https://github.com/marcoabrate/magma.git
```

Then, browse the cloned repository and create a new virtual environment:

```
1  cd magma
2  conda create --name your_env_name --file spec-file.txt
3  conda activate your_env_name
```

At this point, the Transformers library must be installed. A personalized branch is available on my GitHub account. Browse out of the current directory and install it in a new folder:

```
1  cd ..
2  git clone https://github.com/marcoabrate/transformers.git --depth 1 -b
     fine-tuning
3  cd transformers
4  pip install -e .
```

A list of the installed libraries, along with their versions, can be found in the `env.yml` file in the `magma` repository.

C Fine-tuning Parameters

	T5	BART	PEGASUS
GAS	16	16	64
Learning rate	5e-5	5e-5	5e-5
Learning rate scheduler	Constant	Linear decay	Linear decay
Optimizer	Adafactor	AdamW	Adafactor
Freezed embeddings	Yes	Yes	Yes
Label smoothing	0.1	0.1	0.1
Warmup steps	0	0	0
Sortish sampler	Yes	Yes	Yes
Max source length	512	1024	990
Max target length	150	150	150
Val max target length	150	150	150
Steps	∼ 700	∼ 700	∼ 700
Batch size	2	2	2
Evaluation batch size	8	8	8
Evaluation num_beams	2	2	2

D Test Summary Examples

Five random summarization examples picked from the test split of the Merged Overlaps Bullet-Paragraph dataset and generated by the best fine-tuned checkpoint of PEGASUS along with the reference summary and the evaluation metrics.