Bachelor's Thesis Computing Science

Radboud University Nijmegen

Summarization for Long-Form Documents

Leveraging Transformer Models for U.S. Patents Summarization

Author:
Andrei-Sorin Ujica
s1102725

First supervisor/assessor:
Prof. Dr. Ir. David van Leeuwen

Second assessor:
Dr. Harrie Oosterhuis

June 14, 2024

Abstract

This thesis investigates the application of transformers for summarization of lengthy documents, specifically U.S. patents from the `big_patent` dataset and introduces a method called Summary of Summaries (SOS) which is shown to improve summary quality over trivial truncation. The central research question aims to identify the best out-of-the-box model for summarization of U.S. patents and how this model can be fine-tuned for even better results. Key findings include that models optimized for longer sequences, such as BigBird and Long-T5, demonstrate superior performance, with Pegasus-Large achieving a ROUGE-1 score of 0.3245, without any task-specific fine-tuning. Fine-tuning LED improved the performance drastically compared to the pre-trained model, with ROUGE-L increasing from 0.0485 to 0.2430 after only three epochs of training. Incorporating global attention, even on just the first token, increases the ROUGE-LSum score by 0.03 points, resulting in better summary quality.

Contents

Chapter 1

Introduction

The transformer architecture, first proposed in the 2017 paper by Vaswani et al. [20], has been one of the most impactful ideas in recent years in the field of Deep Learning. Before its rise, different neural network architectures like the RNN [5, 18], GRU [6] or LSTM [9] were used for different modalities - text, images, audio, but recently there has been a convergence towards a single architecture - the transformer. This is largely due to the fact that, theoretically, given enough compute resources, the transformer's attention mechanism has an infinite reference window - which means that all input tokens are taken into consideration, improving contextual understanding and enabling parallelizing computations.

One of the tasks where transformers shine are Natural Language Processing (NLP) tasks, and by extension, text summarization tasks. This task can either be extractive - selecting a subset of words from the original document in order to make a summary, or abstractive - paraphrasing the content while capturing the key ideas. While the former was viable using LSTMs, it produced summaries that felt mechanical, not very human-like. The latter can be achieved by leveraging transformer architectures, as their attention mechanism is able to weigh the importance of all the words in the context. [15]

U.S. Patents are a form of intellectual property protection granted for novel inventions that give the holder exclusive rights to the selling and making of the object of that patent. These are usually lengthy documents that follow a standardized format and contain technical jargon. Therefore, they are usually challenging to understand by the general public and would greatly benefit from summarization. Most models targeted at summarization are fine-tuned on standard datasets like the CNN Daily Mail, which may not align well with the specific structure and complexities that U.S. patents have.

This thesis explores the application of transformer architectures in the con-

text of abstractive summarization. It aims to do that by performing a comparative analysis of the most used models on Hugging Face, the leading platform for open-source transformer models. This analysis will take into consideration both quantitative metrics like BLEU, ROUGE, and METEOR and qualitative metrics like human evaluation. Then, it follows by fine-tuning one of the models on a U.S. Patents dataset, optimizing for the specialized language and structure that they have.

1.1 Research Question

Central to this paper is the question: Which transformer model performs the best summarization of U.S. Patents and how effectively can this model be adapted and fine-tuned for even better results for this specific dataset?

1.2 Structure of the Thesis

- **Chapter 2: Preliminaries** explains the transformer architecture, Large Language Models (LLMs), and fine-tuning, among other aspects.
- **Chapter 3: Related Work** assesses existing papers, identifying gaps regarding summarization on datasets related to U.S. patents.
- **Chapter 4: Research** outlines how the model comparison and subsequent fine-tuning was performed.
- **Chapter 5: Results** describes the outcomes of the experiments.
- **Chapter 6: Conclusions** summarizes the paper's findings.

Chapter 2

Preliminaries

This chapter provides the essential background needed in order to understand the rest of this thesis. It covers the basics of deep learning, neural networks, and the transformer architecture. Key concepts such as large-language models (LLMs), fine-tuning, and text summarization techniques are discussed. Also, evaluation metrics and dataset characteristics relevant to summarization are also presented. By the end of the chapter, the reader should have a solid understanding of the foundations needed in order to grasp the next chapters.

2.1 Deep Learning Basics

Deep Learning is a subset of machine learning that involves training neural networks with many layers, hence the term "deep". These networks can model and understand complex patterns in data and can be visualized as a graph. They consist of an input layer, multiple hidden layers, and an output layer, each with activation functions (functions that, given input and weights, return the output of the node).

There exist multiple types of neural network architectures, depending on the type of task. Feedforward Neural Networks (FNNs) [17] pass information in one direction, from the input layer to the output layer, making them suitable for simple tasks like classification or regression. Convolutional Neural Networks (CNNs) [13] process grid-like data, making them ideal for image and video recognition. Recurrent Neural Networks (RNNs) [5, 18] handle sequential data by maintaining memory of previous inputs, making them good for language modeling tasks, even though they struggle with long-term dependencies in the text. To address this issue, Long Short-Term Memory Networks (LSTMs) [9] retain context over longer sequences, making them good for text generation.

2.2 The Transformer Architecture

Transformers are highly effective for multiple different modalities, due to their self-attention mechanism. In order to better understand how they work, a run-through of the encoder mechanism will be presented based on the schematic 2.1. The decoder process is similar due to having the same layers.

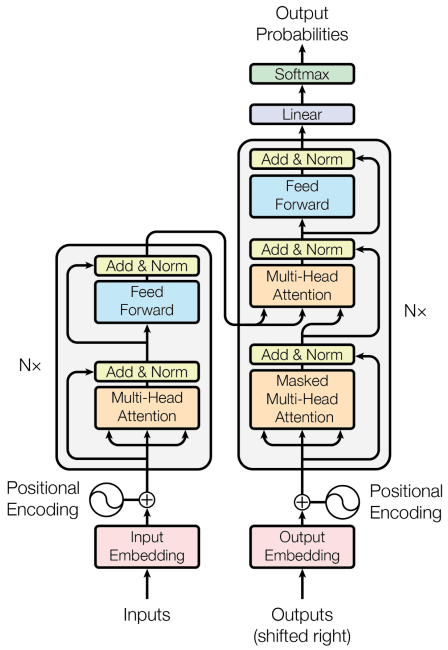

Figure 2.1: Picture taken from the original paper [20]. The transformer architecture.

The first step in generating text with a transformer is feeding the input into a tokenizer. A tokenizer splits the input text into smaller units called tokens (words, subwords, characters). The input embedding layer then converts each token into numerical vectors of fixed size, in order to turn them into representations that the model can process. The next step is to inject positional information about each token into their embeddings. This is done using the sine and cosine functions.

Next, the data is fed into the Encoder Layer. The Encoder layer is responsible for adding context to the embedding. It consists of two submodules: Multi-Head Attention followed by a Feed Forward Network. The former is depicted in figure 2.2. Multi-headed Attention in the encoder applies a specific mechanism called Self-Attention. Self-Attention allows it to associate each individual word in the input with other words in the input.

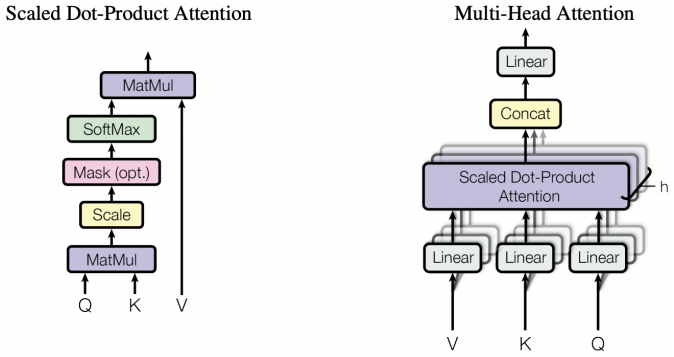

Figure 2.2: Picture taken from the original paper [20]. The Multi-Head Attention Mechanism.

To achieve Self-Attention, the input is fed through three distinct layers to create the query, key and value vectors. These terms come from retrieval systems (like searching for videos) where a query is mapped against a set of keys (video title) in order to return a value (video). In the context of Self-Attention, the queries and keys undergo matrix multiplication in order to produce a score matrix. This score matrix determines how much focus a word should put on other words. Then, these scores are ran through a softmax function, which returns probability values between 0 and 1. Afterwards, the attention weights are multiplied by the values in order to generate the output.

The Multi-Headed Attention output vector is then added to the original input. This is called a residual connection. The residual output gets fed into a feed-forward network for further processing in order to enrich the representation. This wraps up the encoder layer, whose sole purpose is to encode the input to a continuous representation with attention information.

Proceeding to the Decoder Layer, its primary function is to generate text sequences. It has similar layers as the Encoder - two Multi-Headed Attention layers and a Feed-Forward layer, with residual connections after each sublayer. The Decoder is auto-regressive - it takes in a list of previous outputs as inputs, as well as the encoder outputs that contain the attention information. The Decoder stops decoding when it generates an $< end >$ token as an output.

2.3 Large Language Models (LLMs)

Large Language Models (LLMs) are advanced neural networks designed to understand and generate human language. They consist of millions to bil-

lions of parameters (weights and biases that the model adjusts to learn from data), enabling them to capture complex patterns, which is why they are called "large". LLMs can be pre-trained and then subsequently fine-tuned for specific purposes. Pre-training involves training the model on large amounts of data in order to learn general language features for a task, like summarization or classification. Fine-tuning then adapts this model to a niche task, like summarization for patents, using a smaller, task-specific dataset. Examples of LLMs include GPT [4], BERT [7], T5 [16]. In order to limit computational complexity, the input length is limited to a certain number of tokens, usually between 512 tokens and 32K tokens, which might make usage with longer texts harder.

Fine-tuning adapts LLMs to specific tasks by adjusting several key parameters:

- **Learning Rate** - Controls how fast or slow the model learns. A small value like 0.001 helps the model learn steadily, without mistakes.
- **Batch Size** - The number of samples the model processes at once. A higher batch size, like 32, speeds up training but needs more memory.
- **Number of Epochs** - How many times the model goes through the entire dataset. A higher number might lead to overfitting, while a small number might lead to not enough data being processed in order for the model to learn.
- **Gradient Accumulation** - This technique allows the model to handle larger batches by spreading it over multiple steps. For example, with a batch size of 4 and accumulating gradients over 8 steps, it acts like a batch size of 32.

2.4 Text Summarization Techniques

Summarization aims to turn large pieces of text into shorter pieces, while still preserving key information. There are two main approaches: extractive and abstractive [19]. Extractive summarization consists of combining existing sentences from the text while mainintaing the original wording. This might sometimes lead to summaries that lack cohesion and sound robotic. Abstractive summarization, on the other hand, generates new sentences that keep the same ideas from the original text, similar to how a human would summarize. The problem with abstractive summarization is that it is more computationally intensive, and can sometimes generate grammatically or semantically inaccurate sentences.

2.5 Evaluation Metrics for Summarization

Evaluation Metrics assess the quality of the generated summary against a given reference summary. The ones most commonly used are ROUGE, BLEU, and METEOR. All of these give back scores between 0 and 1, where 0 indicates no match between the generated summary and the reference summary, and 1 means a perfect match. An example will be explained for ROUGE as it contains a lot of the fundamental ideas used by the other metrics as well.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [11] measures the overlap of n-grams between the generated summary and the reference summary. There are multiple variants of this: ROUGE-1, ROUGE-2, etc. (collectively called ROUGE-N), and ROUGE-L (which measures the longest sequence of matching words between the reference and generated summary). N-grams represent contiguous sequences of n items (words) from a given text. The example below explains the computation:

Reference Summary: "Police killed the gunman with a knife."
Generated Summary: "Police killed the gunman."

ROUGE-1 (unigram) and ROUGE-2 (bigram) scores are calculated as follows:

Unigrams:

- Reference: {"Police", "killed", "the", "gunman", "with", "a", "knife"}
- Generated: {"Police", "killed", "the", "gunman"}

$$\text{ROUGE-1} = \frac{\text{Number of overlapping unigrams}}{\text{Total unigrams in reference summary}} = \frac{4}{7} \approx 0.571$$

Bigrams:

- Reference: {"Police killed", "killed the", "the gunman", "gunman with", "with a", "a knife"}
- Generated: {"Police killed", "killed the", "the gunman"}

$$\text{ROUGE-2} = \frac{\text{Number of overlapping bigrams}}{\text{Total bigrams in reference summary}} = \frac{3}{6} = 0.5$$

ROUGE measures recall, which measures how many of the relevant words were generated. With generated summaries that are significantly longer than the reference summary, this can become misleading as the score increases but the summary contains information that is not relevant. BLEU (Bilingual Evaluation Understudy) [14] on the other hand, measures precision, which is calculated as the number of matching n-grams divided by the total number of n-grams in the generated sentence. This helps in understanding how much of the generated summary is actually relevant.

METEOR (Metric for Evaluation of Translation with Explicit ORdering) [2] encompasses both precision and recall, then it computes the harmonic mean of these in order to come up with an F1-Score, which is the final reported score of this metric. Also, it takes into account synonyms and stem (base versions of the words) matching in order to get a more comprehensive evaluation.

Summary

The Preliminaries chapter introduces essential background knowledge necessary for understanding the thesis. It starts with a short overview of deep learning and the existing types of neural network architectures, followed by an in-depth look of the transformer architecture. It then continues with Large Language Models (LLMs), describing how pre-training and fine-tuning differ. Finally, the chapter concludes with an explanation of the difference between extractive and abstractive summarization techniques and an example computation of ROUGE-1 and ROUGE-2.

Chapter 3

Related Work

An extensive literature review is necessary in order to understand the current state of research, what the existing gaps in the literature are, and how these gaps can be bridged by the current paper. This chapter is structured into three main sections. First, papers that relate to summarization in general are discussed in order to understand the enhancements that these provide over regular models. Then, an overview of how BigBird [21] and Longformer [3] handle attention is presented. Finally, evaluation metrics such as BLEU, ROUGE, and METEOR are discussed.

3.1 Extractive & Abstractive Summarization

3.1.1 Current State

The application of Transformer models for summarization has advanced significantly since their introduction in 2017 by Vaswani et al. [20], leveraging the architecture's ability to capture long-range dependencies. Transformers use self-attention mechanisms in order to weigh the importance of different words in a sentence, which allows them to handle the relationships between words better, compared to previous architectures like the Recurrent Neural Networks (RNNs) [5], which process sequences sequentially. [20]

Recent studies have shown that transformer models such as BERT [7] or GPT [4] excel in summarization tasks due to their pre-training on large datasets, followed by task-specific fine-tuning. These models achieve superior results on metrics such as ROUGE. For example, a pre-trained Transformer achieved a ROUGE-2 score of 0.131, using only 1% of the training data (approx. 3000 samples), while pre-trained encoder-decoder models scored only 0.023 ROUGE-2 under similar conditions. [10]

Liu and Lapata (2019) [12] introduced a document-level encoder based on BERT for both extractive and abstractive summarization. In this approach,

the BERT [7] model first processes individual sentences, generating representations that also contain the context (can be seen on the first line in figure 3.1). Then, additional transformer layers are stacked on top of these sentences in order to capture the different relations across the document. This is useful for maintaining logical flow in the summary as it helps the model understand how the sentences relate to each other in the context of the entire document.

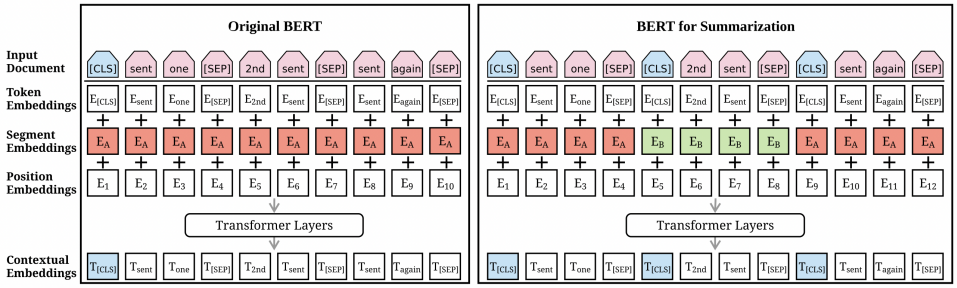

Figure 3.1: Image from the original paper [12]. It shows a comparison between BERT(left) and BERTSum(right). BERTSum inserts CLS tokens after every separator token in order to segment the input into sentences.

For abstractive summarization, Liu and Lapata (2019) [12] proposed a new fine-tuning schedule that uses two different optimizers for the encoder and decoder, each with its own warmup steps and learning rates. Since the encoder is pretrained and the decoder must be trained from scratch, this results in a mismatch. For example, the encoder might overfit on data while the decoder underfits, or vice versa. Using two different optimizers solves this problem. Additionally, a two-step fine-tuning process is presented: The first step consists of extractive summarization in order to get the most important sentences, followed by a second step where abstractive summarization is employed in order to make the summary more coherent and logically connected.

3.1.2 Gaps in the Literature

Despite the progress, several gaps remain in transformer-based summarization. First of all, extractive summarization, while able to identify key sentences, results in summaries that are not fully coherent and readable. Also, the ability of extractive models to generalize across different domains is limited, with their performance depending heavily on the structure and writing style of the training data. [15]

Another gap is the scalability of transformers for long documents. Transformers use self-attention, where each token relates to all other tokens. [20].

Therefore, the runtime increases quadratically with the size of the input, limiting their applicability to longer texts. In other words, for an input of size N, the time complexity of the self-attention mechanism is $O(N^2)$, which is sub-optimal.

3.2 Long-Document Summarization

3.2.1 Current State

Summarizing long documents poses problems due to the context that needs to be maintained across all sentences. Traditional sequence-to-sequence models struggle with long documents because their memory usage increases quadratically with the input length. Recent advancements in long-document summarization address these issues, with solutions such as combining extractive and abstractive techniques or changing the window across which attention is computed.

Pilault et al. (2020) [15] proposed a hybrid approach for summarizing long documents by combining extractive and abstractive summarization. The method begins with an extractive step, which selects the sentences that have the most salient (relevant for the summary) content. Then, an abstractive summarizer rewrites the previously generated summary into the final summary. This process reduces the computational resources needed while also ensuring that only the most relevant content is included. This approach achieved higher ROUGE-1, ROUGE-2, and ROUGE-L scores compared to other methods, achieving 0.396, 0.121, and 0.357 respectively on the arXiv dataset. [15]

BigBird, introduced by Zaheer et al. (2020) [21] addresses the issue of the increased computational complexity for large inputs by introducing a sparse attention mechanism (visually explained in figure 3.2). Instead of attending to every single token in the sequence, BigBird combines local attention - which restricts the tokens that one other token can relate with to a predefined window; global attention - which lets one token relate to all the others and random attention - where some tokens relate to a random number of tokens, at random positions in the input. This reduces the quadratic complexity to linear complexity, making it feasible to process long documents.

Beltagy et al. (2020) [3] proposed Longformer, another transformer model that is optimized for long documents. Similarly to BigBird, it addresses the problem of the quadratic time complexity by changing the attention mechanism. A new model is introduced, the Longformer Encoder-Decoder (LED), which uses both global and local attention. This model has both the encoder and the decoder from the original Transformer architecture [20]

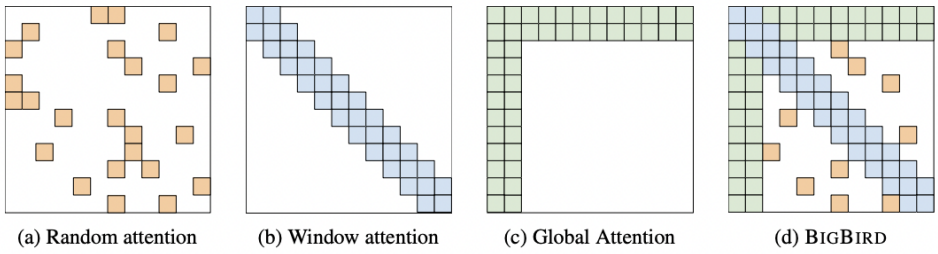

Figure 3.2: Image from the original paper [21]. It shows a comparison between different types of attention. White color represents no attention.

but instead of having full self-attention in the encoder, it uses the sparse attention mechanism described earlier. The decoder maintains the full self-attention. The LED parameters are initialized from BART, therefore, in order to process 16K token inputs, the 1K positional embeddings of BART have been copied over 16 times. [3]

3.2.2 Gaps in the Literature

Even though the techniques introduced here address the problem of long document inputs, one gap still remains: the ability of these models to generalize across multiple domains. As most summarization models are trained on news datasets, which are highly extractive due to having most of the salient content (content that is relevant to the summary) in the beginning, the model does not learn any patterns that can be transferred over to other domains. [19]

3.3 ROUGE, BLEU & METEOR

3.3.1 Current State

In order to evaluate the quality of generated summaries, evaluation metrics are crucial. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is widely used for comparing the overlap of n-grams between generated summaries and reference summaries. [11]. The paper presents multiple variants: ROUGE-1 (measures unigram overlap), ROUGE-N (measures n-gram overlap), ROUGE-L (measures the longest common subsequence). One metric commonly used for summarization is ROUGE-LSum (which is also called Union LCS), which splits the summary into sentences, then computes ROUGE-L for each pair of sentences, finally averaging them out for a final score. In this way, ROUGE-LSum penalizes differences in sentence structure more than ROUGE-L. [11]

BLEU (Bilingual Evaluation Understudy) [14] and METEOR (Metric for Evaluation of Translation with Explicit ORdering) [2] are also used in summarization evaluation. BLEU has a similar approach to ROUGE, as it also counts matching n-grams, but BLEU looks at precision (what percentage of the generated n-grams are matching n-grams). One other difference between them is that BLEU applies a brevity penalty for shorter summaries, which might end up prioritizing precision over recall [14]. METEOR fixes some of BLEU's limitations by incorporating synonym matching. Therefore, it depicts the quality of both abstractive and extractive summaries more accurately. Also, METEOR uses stemming, the process of reducing words to their root form (for example, "running" becomes "run"). This approach helps recognize different forms of the same word.

3.3.2 Gaps in the Literature

ROUGE, BLEU, and METEOR primarily focus on n-gram overlap and might not accurately represent the fluency of a generated summary. Instead, these metrics reward verbatim copy-pasting over abstractive summarization, missing the nuances of the text.

Summary

To summarize, transformer models have improved at the task of summarization, due to their self-attention mechanism capturing long-range dependencies better than RNNs. Models like BERT and GPT are very good due to their large training sets, achieving high ROUGE scores and improved coherence. However, they struggle with coherence in extractive summaries and generalization as they are usually fine-tuned on news datasets, which are highly extractive. This thesis contributes to addressing these gaps by fine-tuning Longformer Encoder-Decoder (LED) on the `big_patent` dataset, which is inherently more abstractive and has the salient content better distributed throughout the data. Therefore, this enhances the existing understanding related to improving summary coherence and recall.

Chapter 4

Research

This chapter delves into the methodology used to measure the current capabilities of state-of-the-art summarization models and, subsequently, the effects of fine-tuning on their performance when dealing with datasets characterized by longer inputs, such as `big_patent`. This analysis is structured into two principal sub-chapters: the first one evaluates various models in their 'out-of-the-box' state, in order to gain a better understanding of their baseline capabilities; the second one focuses on the fine-tuning of the Longformer Encoder-Decoder on the `big_patent` dataset, chosen due to its large context, suitable for the description lengths that will be fed into the model. The findings from these studies will be examined in-depth in the Results chapter, providing insights into both the general applicability of these models to long-document summarization and the advantages of fine-tuning them for this specific task.

4.1 Benchmarking

4.1.1 Model Selection

In order to choose the appropriate models for the benchmarking, the dataset was analyzed. With description lengths between 150 and 80.000 words [19], it required models capable of handling long input sequences. Therefore, models with large input context windows were prioritized, such as `BigBird`, `Long T5`, and `LED`, which can manage input lengths up to 16.384 tokens.

Afterwards, various model sizes were included in order to ensure a comprehensive range of capabilities. Smaller models like `t5-base` (223M parameters) and `bart-base` (139M parameters) were added to evaluate their performance in contrast to the larger models.

Finally, both fine-tuned and base versions were selected to assess the impact of task-specific fine-tuning on performance and efficiency. This helps

determine if fine-tuned models significantly outperform the base ones in summarization tasks.

Model Name	Max Input	Fine-Tuned?	No. of params.
t5-base	512 tokens	No	223M
bart-base	1024 tokens	No	139M
pegasus-large	1024 tokens	No	568M
bigbird-pegasus-large-arxiv	4096 tokens	Yes	568M
long-t5-tglobal-base-sci-simplify	16384 tokens	Yes	248M
led-base-16384	16384 tokens	No	406M

Table 4.1: Overview of Selected Models

4.1.2 Dataset Description

The **BigPatent** dataset is an extensive collection of over 1.3 million U.S. patent documents, curated for research into summarization. It comprises of nine different categories, labeled A – H and Y. In our benchmarking and fine-tuning research, we are only interested in section G (Physics), solely based on its dissimilarity to the usual news summarization datasets. The data in category G is split into train, validation and test splits as follows:

- Training Set: 146K rows
- Validation Set: 14.4K rows
- Test Set: 14.4K rows

Most existing summarization datasets are comprised of news articles, where the summary-worthy content is usually in the beginning. Moreover, in these datasets, large segments of text are usually present word for word in the summary, a characteristic called low abstractiveness. These issues impede the learning of models meant to understand the article's global structure as well as produce highly abstractive summaries.

Summary-worthy content, as referred to in the last paragraph, is called salient content in the literature. Salient content distribution is one of the strong characteristics of this dataset. In popular summarization datasets based on news articles such as `CNN/Daily Mail` or `Newsroom`, this content is concentrated in the beginning. In **BigPatent** documents, on the other hand, this content is distributed throughout the entire document - figure 4.1 gives an intuition on this matter. This means that, to generate a complete summary, a model would need to understand the entire text, not just the beginning.

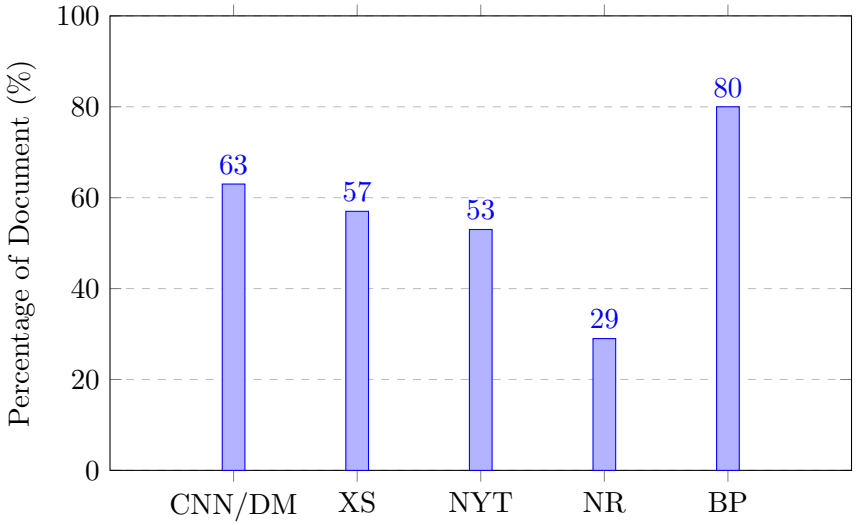

Figure 4.1: Percentage of the document required to cover all salient words for different datasets for CNN/DM(CNN/Daily Mail), XS(XSum), NYT(NYT), NR(Newsroom), BP(BigPatent) [19]

An n-gram is a sequence of n items from a text or speech. For example, in the phrase "This dataset is big." some bigrams (2-grams) are "This dataset", "dataset is", etc. Abstractiveness measures the number of new n-grams that do not appear in the input text. **BigPatent** documents are highly abstractive, containing a higher percentage of new n-grams compared to other datasets. [19]

Therefore, to sum up, this dataset was chosen for its uniform distribution of salient content and high abstractiveness, which require models to understand the global context and generate new summaries.

4.1.3 Evaluation Metrics

To measure the quality of the summaries several evaluation metrics were used: ROUGE, BLEU and METEOR.

ROUGE has been chosen because it effectively measures the overlap of n-grams, which is important when trying to determine how much information from the reference summary has been captured in the generated summary. BLEU is included because it measures precision, which is particularly important for models with smaller contexts, like `t5-base`. Precision refers to

the percentage of matching n-grams that are present in the generated summary, ensuring that even shorter outputs are represented fairly. METEOR was selected because, unlike BLEU and ROUGE which measure the syntactic similarity, measures the semantic similarity, accounting for variations in wording or structure.

4.1.4 Experimental Setup

This section will describe the procedure used in conducting the benchmark. It is implementation-agnostic, as it should be reproducible by anyone reading this paper; for specific implementation details, please refer to the Appendix.

The procedure begins with loading the dataset, more specifically the `test` split of the `g` category. The reasoning here is that this split consists of data that will not be encountered during training, so it's a more accurate measure of how it will perform on real-world scenarios.

Then, once a model is chosen, it is loaded onto the GPU if available, otherwise onto the CPU. The corresponding tokenizer for the selected model is also initialized. A tokenizer's role is to pre-process and transform the input data from a string to a list of numerical representations, to be then fed into the model. During tokenization, the input is padded to the required length of each model. If the input exceeds the model's maximum token length, the overflow is handled by returning two chunks: the first one contains the tokens that fit, the other one contains the tokens that did not. If the tokenizer returned two chunks, the approach detailed in 4.1.4 will be performed.

Afterwards, the summarization step begins. Due to the nature of the dataset, with inputs between 150 and 80.000 words [19], chunking of the input is often required.

Chunking involves breaking down the text into smaller, more manageable pieces that fit inside the model context window; each of these chunks have an overlap of 16 tokens with each other so that the context is preserved. If the input fits into a single chunk, it is directly passed to the model. Otherwise, if the input spans multiple chunks, each one is summarized individually. These chunks are then concatenated and summarized again to produce the final summary. In the following chapter, this method will be called "Summary of Summaries" or SOS, in short.

Finally, once inference finishes running, the prediction and its corresponding reference are stored. Once every document in the dataset has been processed, the evaluation metrics - ROUGE, BLEU and METEOR, are computed using the predictions and references arrays.

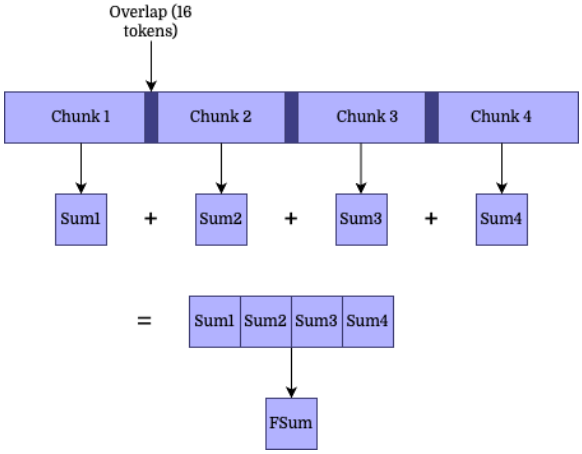

Figure 4.2: Chunking process with visible overlaps. Each chunk is then concatenated, tokenized and fed through the model again.

4.1.5 Summary

This benchmarking subchapter outlined the process of evaluating various summarization models on the **BigPatent** dataset. It began by justifying the model selection, which was largely influenced by the maximum input length of the models. Then, the dataset characteristics have been discussed, arguing that salient content distribution and abstractiveness are two of the main advantages of using **BigPatent** over a regular new summarization dataset. Afterwards, the choice of evaluation metrics was explained - to develop a balanced evaluation framework which would measure both syntactic and semantic similarity. Finally, the experimental setup was presented, describing the chunking mechanism used in order to fit the description of the patents into the selected models.

4.2 Fine-tuning

4.2.1 Dataset Preparation

In order to preprocess the data effectively, it is important to understand the distribution of the token lengths within the descriptions and the summaries. This can be achieved by taking a sample from the validation set, tokenizing it and finally generating a histogram of the tokenized lengths in order to better visualize the distribution. To determine the number of bins, the Freedman-Diaconis rule was applied, which considers the interquartile range and the number of data points, helping ensure an accurate representation. [8] The histograms reveal that the mean and median lengths of the descriptions were approximately 10,000 tokens, while the summaries aver-

aged around 100 tokens (figure 4.3), which will be helpful in setting up the padding and truncation strategies.

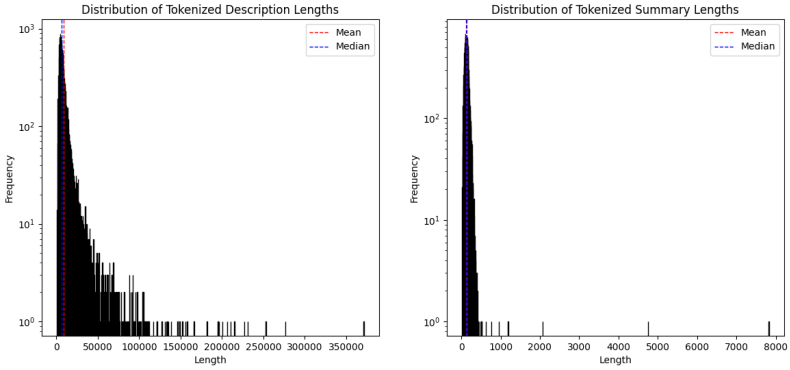

Figure 4.3: Length distribution of a 10.000 rows sample of the validation split. The mean and median are displayed.

Now that there is a clearer understanding of the lengths of both descriptions and inputs, a chunking strategy can be established for the ones that exceed LED's 16K tokens input. The approach used involves using an NLP library in order to split the summary into sentences. If a summary contains fewer than two sentences, a default value of five is used. This was done in order to ensure that the input chunks that result are not too large for the model.

For the input descriptions, the text was divided into n chunks, where n represents the number of sentences in the corresponding summary. This approach ensures that the mappings of the description chunks and summary chunks are proportionally aligned, which ensures that the context is not lost. If the chunk size is larger than 16K tokens, it is truncated. This chunking strategy was applied uniformly to both the training split and the validation split, essentially transforming a single description and its summary into multiple smaller, aligned parts.

Afterwards, each chunk was tokenized and a data collator was employed. As the data is processed in batches, the data collator finds the maximum length within each batch and pads all the input sequences and their corresponding labels to that length, ensuring uniformity in the input of the model.

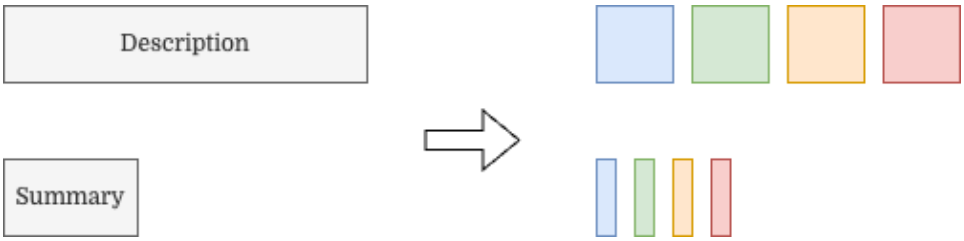

Figure 4.4: The visualization of the chunking process. The mapping between the chunks are represented by the different colorings.

4.2.2 Hyperparameter Optimization with Optuna

Optuna is a hyperparameter optimization framework designed to automate the search for optimal hyperparameters for model training - it involves running multiple trials and evaluating which one performs best. In order to decide which hyperparameters to try next, it leverages a technique called Tree-Structured Parzen Estimator (TPE), which involves two big steps.

First, as the trials are completed, Optuna builds a probabilistic model of the objective function based on the results of the previous trials which estimates the likelihood of different hyperparameters achieving good performance. Then, it uses an acquisition function in order to balance exploration (trying new, less certain parameters) and exploitation (focusing on parameters that have performed well in the past). This is repeated for each trial run in order to evaluate different hyperparameter combinations. [1]

In this study, several key hyperparameters were targeted for optimization: learning rate, number of training epochs, batch size, and gradient accumulation steps. First, the batch size and gradient accumulation steps were manually increased in order to find the biggest effective batch size (batch size $\times$ gradient accumulation steps) - this is compute-dependent. Then, the learning rate was varied between 1×10^{-5} and 5×10^{-5} on a logarithmic scale. The number of epochs was set between 3 and 6, in order to balance training time and compute power against overfitting. Mixed precision was used (fp16) - using 16-bit floating-point types as well as the regular 32-bit, in order to speed up computation, which is particularly useful for large models such as LED.

The hyperparameter optimization study was conducted with the objective of maximizing the ROUGE-LSum score, a metric commonly used to evaluate the quality of generated text by measuring the longest common subsequence between the reference and generated summaries. The study used Optuna's pruning capabilities to terminate unpromising trials early, based on inter-

mediate evaluation losses, thus conserving computational resources.

Trial	ROUGE-LSum	Learning Rate	Number of Epochs
0	0.1839	1.44e-05	3
1	0.2022	4.89e-05	3
2	0.1931	4.33e-05	5
3	0.1980	4.50e-05	4
4	0.1931	1.38e-05	4
5,6,7,8,9	Pruned	-	-

Table 4.2: Results of Hyperparameter Optimization Trials

The optimization results indicate that the highest ROUGE-LSum was achieved with a learning rate of 4.89e-05 over three epochs - 0.2022. Due to compute and cost limitations, other hyperparameters were manually set. For example, mixed precision float type (both 16-bit and 32-bit representations) was used in order to reduce computational usage.

4.2.3 Training Process

The training process for fine-tuning began with the initialization of the model using pre-trained weights from the `allenai/led-base-16384` model. The training and validation datasets were loaded, specifically the `g` split. A subset of 9000 samples from the training set and 1000 samples from the validation set were randomly selected by shuffling the dataset in order to manage the computational load. Each sample was further split into approximately 6 chunks on average, effectively increasing the number of training examples to around 50.000 for the training set and 5000 for the validation set. The training loop involved iterating over the dataset for three epochs, with each epoch involving forward and backward passes to update the model parameters based on the computed loss.

The following hyperparameters were used during training:

- optimizer: Adam with betas=(0.9,0.999) and epsilon=1e-08
- gradient_accumulation_steps: 2
- learning_rate: 4.89e-05
- learning_rate_scheduler_type: linear decay
- train_batch_size: 2
- eval_batch_size: 8

- num_epochs: 3

Evaluation was an important part of the training process, as it helped monitor the model's performance and ensure it was not overfitting to the training data. The evaluation occurred at the end of every epoch on the validation split, with ROUGE-LSum being the primary metric used. ROUGE-LSum is a variant of ROUGE, in which ROUGE-L is computed at the sentence level, taking the longest sequence of words that match. Then, the sentence-level scores are averages in order to get the final score. This is more accurate for summarization as it penalizes for a structure that is different to the reference structure.

Training Loss	**Epoch**	**Validation Loss**	**Rouge1**	**RougeLSum**	**Generated Length**
0.2195	1.0	0.2350	0.2858	0.2370	50.5544
0.1754	2.0	0.2363	0.2895	0.2389	49.5847
0.1428	3.0	0.2412	0.2972	0.2430	46.3260

Table 4.3: Results of Evaluation During Training. Generated Length represents the length, in tokens, of the generated summary.

As shown in table 4.3, the training loss has decreased consistently across the epochs, indicating that the model was effectively learning from the training data. This is a positive sign, showing that the model's predictions have become more accurate. While the validation loss remaining relatively stable is not optimal, as it should have ideally decreased along with the training loss and then plateaued, the ROUGE scores increasing and the generation length decreasing are positive signs. This reflects that the summaries are becoming more concise and potentially more accurate.

4.2.4 Summary

This subchapter details the preprocessing, hyperparameter optimization and training of the Longformer Encoder-Decoder (LED) model. Token length distributions are computed - the mean is 10.000 tokens for the descriptions and 100 tokens for the summaries. Then, the chunking algorithm is described - each summary is split into ‘n‘ sentences and then the description is split into ‘n‘ chunks. After performing a study using Optuna, the model is fine-tuned with the resulting hyperparameters, showing decreasing training loss and increasing ROUGE scores, with a final ROUGE-LSum score of 0.2022, indicating effective model training.

Chapter 5

Results

This chapter presents the results of multiple experiments to evaluate and improve LED's performance on long-form documents. It starts with an analysis of the benchmarking results from Chapter 4. Then, it continues with a look into how different prompts influence the output quality of the model, by looking at standard prompts, instruction-based prompts and example-driven prompts. Next, we examine the impact of manipulating the length parameter at inference-time, and how this relates to ROUGE. Finally, we investigate the role of global attention in handling long-context documents.

5.1 Benchmarking Results

The performance of multiple models was evaluated on a sample from the test split using ROUGE, BLEU and METEOR. Figure 5.1 presents the summarized results for each model. These results highlight several factors that influence the summarization performance like model size and input length capacity, to name a few.

Model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU	METEOR
T5-base	0.0897	0.0212	0.0737	0.0038	0.0400
BART-base	0.0442	0.0083	0.0396	0.0002	0.0172
Pegasus-large	0.3245	0.0883	0.1904	0.6396	0.1939
BigBird-Pegasus	0.2183	0.0362	0.1410	0.6689	0.1412
Long-T5	0.2821	0.0388	0.1529	0.8348	0.2030
LED-base	0.0563	0.0112	0.0486	0.0387	0.0292

Table 5.1: Benchmark Results on the `big_patent` Dataset

Firstly, it is obvious that larger models (refer to table 4.1.1) perform better on long-document summarization tasks. For example, Pegasus-large and Long-T5 both outperformed smaller models like T5-base and BART-base.

This is most likely due to the larger number of parameters in the first models, allowing them to capture more complex patterns and dependencies. Specifically, Pegasus-large achieved a ROUGE-1 score of 0.3245, significantly higher than the 0.0897 of T5-base.

Secondly, the ability to handle longer input sequences (refer to table 4.1.1) seems to be of great importance for extensive documents. Models designed to process longer sequences, such as BigBird-Pegasus and Long-T5, had superior performance compared to those with shorter input contexts. For example, BigBird-Pegasus, with an input length of 4096 tokens, achieved a ROUGE-1 score of 0.2183, whereas BART-base, limited to 1024 tokens, only scored 0.0442. This highlights the importance of optimizing for long contexts, therefore ensuring that the input is not truncated and that the generated summary is complete and relevant.

Lastly, the degree of fine-tuning and the model objectives impact the performance of summarization. Pegasus-large, with its specialized summarization training, outperformed other models, indicating that models pre-trained on tasks that closely align to the target task perform better. On the other hand, the LED-base model, despite having the capacity to handle long inputs (16384 tokens), achieved lower scores, suggesting that further fine-tuning on summarization tasks is needed to increase the performance.

5.2 Comparative Analysis of Summary of Summaries Approach

In this experiment, a comparative analysis between a trivial truncation approach and the Summary of Summaries (SOS) approach (refer to subsection 4.1.4) has been conducted in order to understand if the latter brings any improvements to the summarization process. The model used in this experiment is the fine-tuned version of LED presented in the previous chapter.

The analysis has been conducted on a sample from the test split, where the trivial approach truncated each input to 16K tokens, while the SOS approach used a chunk size of 4K tokens and an overlap of 128 tokens. The boxplots above present the distributions of ROUGE, BLEU and METEOR for both methods. All three evaluation metrics have been employed in order to understand both recall and precision.

Looking at the boxplots, it is obvious that the SOS approach scored better on every metric. For ROUGE, scores range from 0.081 to 0.574 for the truncation, while for the SOS approach, they range from 0.137 to 0.500, showing a smaller range and thus more stable results. The mean scores are also

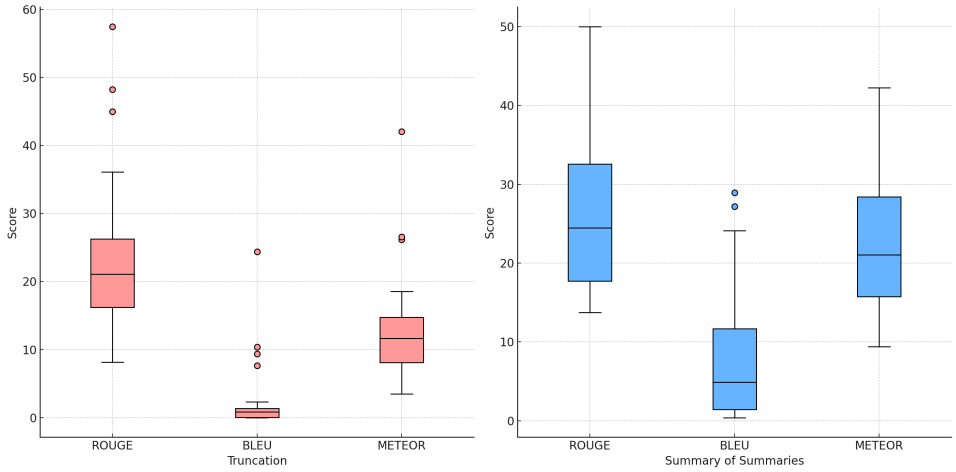

Figure 5.1: Boxplots comparing truncation(left) with summary of summaries(right). All scores were scaled by a factor of 100 to display them better .

higher - 0.276 for SOS compared to 0.236 for truncation. BLEU scores are drastically better in the second approach, with a mean of 0.083 for SOS and 0.023 for truncation. This shows that the truncation method lacks precision, meaning that it generates content that is not relevant to the reference summary. Finally, METEOR scores, the most comprehensive of the three due to it comparing both recall and precision, with synonym matching, also show that the SOS approach performs more qualitative summaries. The mean score for SOS was 0.223, while the mean score for the truncation method was 0.127. Therefore, it is clear that using a two-step approach like SOS helps in generating more relevant summaries (shown by the higher precision) and to also retain the key information (shown by ROUGE and METEOR).

5.3 Prompt engineering

Prompt engineering is the practice of optimizing the prompts that are fed into the model in order to maximize the model's performance. For this experiment, the fine-tuned LED model from the previous chapter is used. Mathematically, given a prompt P, the model generates a response R by sampling from the probability distribution $P(R \mid P)$. The model calculates the probability of each token r_i in the response sequence $r_1, r_2, .., r_n$ based on the prompt P and previously generated tokens.

$$P(R \mid P) = \prod_{i=1}^{n} P(r_i | r_{1:i-1}, P)$$

Changes in the prompt P can significantly influence the values of these

probabilities and therefore change the response R. Every transformer model has a vocabulary - a list of all the words it can use. For each r_i, a raw prediction score (logit) is generated for every single word in the vocabulary. Finally, these are passed through a softmax function in order to make sure all the probabilities sum up to 1. This example showcases how the prediction step works for the basic prompt "The device converts thermal energy":

$$\begin{aligned} P(\text{“into”} \mid \text{“The device converts thermal energy”}) &= 0.7 \\ P(\text{“to”} \mid \text{“The device converts thermal energy”}) &= 0.2 \\ P(\text{“and”} \mid \text{“The device converts thermal energy”}) &= 0.05 \\ P(\text{other tokens} \mid \text{“The device converts thermal energy”}) &= 0.05 \end{aligned}$$

The highest probability is assigned to "into", so the model would likely generate "into" as the next token. In order to evaluate the influence of different prompts on the quality of the output, an experiment was conducted using these four prompt categories:

- **Basic:** "Summarize the following patent:"
- **Contextual:** "Summarize the following patent related to Physics:"
- **Instruction-Based:** "Provide a brief summary highlighting the key innovations and technical details of the following patent:"
- **Example-Driven:** "Here is an example of a good summary: {example}. Now, summarize the following patent in a similar manner:"

For each one, inference was run on a sample from the validation split and the ROUGE-LSum score was computed. The results are plotted on the candlestick chart in figure 5.2.

Quantitatively, the results indicate that, while the prompts influence the result, they do it in a manner that marginally modifies the ROUGE scores. These scores can be split in two categories: a broad interquartile range and a higher median (Basic and Example-Driven) or a narrower interquartile range but a lower median (Contextual and Instruction-Based). The first category shows variability but effective performance, suggesting that providing examples during inference, also known as few-shot inference, improves (even if marginally so) the results of the basic prompt. The second category shows more stability but suggests that adding context alone is not sufficient, but rather detrimental to the quality of the output.

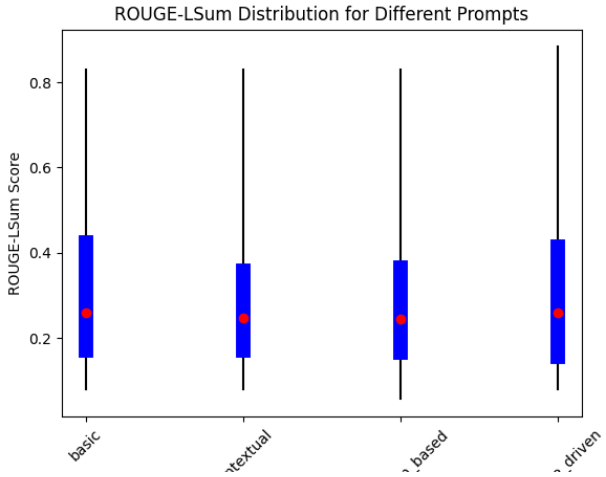

Figure 5.2: Distribution of ROUGE-LSum scores for each prompt type on the fine-tuned LED model. The candlestick body represents Q1 to Q3, the dot represents the median and the line extends to the minimum and maximum values.

5.4 Maximum Length Variation During Inference

When generating summaries with a model, the minimum and maximum length parameters can be modified during inference. Practically, the model will not generate an `<end>` token until it reaches the minimum length, and it will forcefully generate the same token if it goes past the maximum length. This allows for observations into how variation in the length of the output influences the quality. In this experiment, various minimum - maximum length ranges were examined (8-16 tokens all the way up to 128-256 tokens) and ranked based on the ROUGE-LSum score of the generated summaries. The model used in this experiment is the fine-tuned version of LED presented in the previous chapter. Table 5.2 outlines these results.

Length Range	**Q1**	**Median**	**Q3**	**IQR**
8-16 tokens	0.1197	0.1561	0.1730	0.0533
16-32 tokens	0.1623	0.2109	0.2625	0.1002
32-64 tokens	0.2336	0.2810	0.3897	0.1561
64-128 tokens	0.2624	0.3668	0.5709	0.3085
128-256 tokens	0.2606	0.3608	0.6057	0.3451

Table 5.2: ROUGE-LSum Scores for Different Length Parameters. IQR here represents the interquartile range.

Looking at it from a quantitative standpoint, the data indicates that, as the maximum length parameter increases, the ROUGE-LSum score increases as well, meaning that long outputs present better results. For example, the median score rises from 0.15 for the 8-16 tokens range to 0.36 for the 128-256 tokens. This was expected, as the median length of the summaries in the training set is around 150 tokens, which suggests that allowing the model to generate longer text helps with improving contextual understanding.

However, it is important to note that higher ROUGE scores might also be due to the higher number of words generated, meaning that there is a higher likelihood that one of the generated words is in the reference summary. Since ROUGE does not penalize for additional content not in the reference, longer summaries might simply score higher due to their length, which is why there is a need to also perform a qualitative analysis. Despite this, the interquartile range increases from 0.05 to 0.34 as the length of the output increases, which indicates higher variability.

Next, a qualitative analysis will be performed in order to better understand the differences. Here are examples of generated text with 32 and 128 tokens to illustrate the practical impact and compare key ideas:

Reference Summary: "A magnetic resonance imaging method for high-quality short echo time imaging uses FID-acquired-echoes. Gradient recalled echoes are excited in two short echo time acquisition sequences having opposite polarity reading gradients. Half-echoes for corresponding phase encoding steps are concatenated to reconstructed echoes. An image is reconstructed from the reconstructed echoes using a Fourier transform reconstruction method. With the method, artifact-free high-quality images with echo times below 4 msec can be obtained, even under unfavorable conditions, such as fast blood flow."

32-token Output: "The invention relates to a magnetic resonance imaging method, in which an object is situated in a stationary homogeneous magnetic field and is subjected to an acquisition sequence"

- Missing key ideas: FID-acquired echoes, gradient-recalled echoes, artifact-free high-quality images

128-token Output: "The invention relates to a magnetic resonance imaging method in which an object is situated in a stationary homogeneous magnetic field and is subjected to an acquisition sequence for acquiring a magnetic resonance signal from a region of the object, the acquisition sequence comprising an rf-pulse for exciting atomic spins in the object, a phase encoding gradient, and a reading gradient which is reversed for acquiring the

magnetic resonance signal, the acquisition sequence being repeated a number of times with a different value of the time integral of the phase encoding gradient."

- Missing key ideas: artifact-free high-quality images

These examples showcase that the shorter output is incomplete and misses critical details such as FID-acquired echoes and gradient-recalled echoes, while the longer one provides a more detailed version, but still misses the idea of an artifact-free image. This aligns with the quantitative data, demonstrating that maximizing the minimum and maximum length parameters during inference time can enhance the thoroughness of the generated text.

5.5 Global Attention

Attention is a mechanism in transformer models in which each token weighs the importance of all the other tokens. This works well for shorter inputs, but being quadratic in time complexity, it becomes impractical for long documents. This results in increased memory and computational requirements. Therefore, in order to address this problem, the Longformer Encoder-Decoder (LED) uses both local attention and global attention. Local attention restricts each token to attend to a predefined number of neighbors (for LED it is 1024), while global attention allows certain tokens to attend to all the other tokens in the sequence, capturing long-range dependencies more effectively.

An experiment was conducted in order to compare the performance of the fine-tuned version of LED (presented in the previous chapter) using local attention versus using both local and global attention. In this experiment, global attention was applied to the first token of the input, as this token should have the context of the entire text, while in the local attention setup, no tokens had global access. Table 5.3 outlines the results of computing ROUGE-LSum on the generated summaries.

Attention Type	**Q1**	**Median**	**Q3**	**IQR**
Local	0.2699	0.3700	0.5815	0.3116
Global	0.2692	0.3959	0.5713	0.3026

Table 5.3: ROUGE-LSum Scores for Different Attention Types. IQR here represents the interquartile range

These results indicate that using global attention improves the model's performance drastically. Even though global attention was applied only to the

first token, this resulted in an increase from 0.37 to almost 0.40, demonstrating better summary quality. Also, the interquartile range for global attention is lower, indicating less variability with global attention and more consistent results.

Summary

This chapter details the different experiments that were conducted in order to optimize the quality and performance of the LED model fine-tuned on `big_patent`. First, the results of the benchmarking show that models with longer input sequences, more parameters and with model objectives similar to the target task perform better. Then, a look into how prompts influence the results showed that the influence is marginal, with basic and example-driven prompts performing slightly better. Afterwards, it is shown that having longer outputs can improve the ROUGE scores and provide a more complete summary. Finally, global attention is discussed, and it is established that global attention drastically improved the quality of the summary, at the price of extra computational resources.

Chapter 6

Conclusions

This thesis explored the application of transformer models in the context of long-form documents, specifically for U.S. Patents, which have lengths that can go up to 80.000 words. The central research question was to determine which transformer performs the best summarization of U.S. patents and how effectively this model can be adapted and fine-tuned for better results.

6.1 Key Findings

The key findings indicate that larger models, such as `Pegasus-Large` and `Long-T5` outperform smaller ones like `T5`, due to their ability to cover more complex patterns. Also, models optimized for longer sequences, such as `BigBird` and `Longformer Encoder-Decoder(LED)` demonstrated superior performance. Fine-tuning LED on the `BigPatent` dataset improved ROUGE scores compared to the non-finetuned version, although further fine-tuning is necessary for optimal performance. Additionally, adding global attention significantly enhanced the model performance, resulting in better summary quality, as shown by the increasing ROUGE scores.

Implications and Future Work

The findings from this thesis contribute to understanding how transformers can be optimized for summarization of lengthy documents like U.S. Patents. Future work could focus on more advanced fine-tuning methods or additional training using more powerful computing. Also, one other aspect that future papers could focus on is using more powerful evaluation metrics, that better capture the quality of a summary, besides n-gram overlap.

Bibliography

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *arXiv preprint arXiv:1907.10902*, 2019.

[2] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72.

[3] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020.

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[5] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[6] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[8] David Freedman and Persi Diaconis. On the histogram as a density estimator: L2 theory. *Probability Theory and Related Fields*, 57(4):453–476, 1981.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[10] Urvashi Khandelwal, Kevin Clark, Dan Jurafsky, and Lukasz Kaiser. Sample efficient text summarization using a single pre-trained transformer. *CoRR*, abs/1905.08836, 2019.

[11] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[12] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. *CoRR*, abs/1908.08345, 2019.

[13] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.

[14] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.

[15] Jonathan Pilault, Raymond Li, Sandeep Subramanian, and Chris Pal. On extractive and abstractive neural document summarization with transformer language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[16] 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. *CoRR*, abs/1910.10683, 2019.

[17] Murat Sazli. A brief review of feed-forward neural networks. *Communications Faculty Of Science University of Ankara*, 50:11–17, 01 2006.

[18] Robin M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *CoRR*, abs/1912.05911, 2019.

[19] Eva Sharma, Chen Li, and Lu Wang. BIGPATENT: A large-scale dataset for abstractive and coherent summarization. *CoRR*, abs/1906.03741, 2019.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, volume 30, 2017.

[21] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. *CoRR*, abs/2007.14062, 2020.

Appendix A

Implementation Details

This appendix provides an overview of the implementation of the benchmarking and fine-tuning processes for the `LED` model on the `big_patent` dataset. While the full codebase is hosted on GitHub here, the following sections highlight some of the most interesting parts of the implementation.

A.1 Computational Setup

All development was done using Hugging Face Spaces, utilizing an Nvidia N10G with 12 vCPUs, 46GB RAM, and 24GB VRAM. The libraries used include:

- Transformers 4.38.2
- PyTorch 2.2.1+cu121
- Datasets 2.18.0
- Tokenizers 0.15.2

A.2 Benchmarking

In order to run the benchmark, a Hugging Face Space has been created. Gradio is leveraged as a front-end framework, in order to iterate rapidly. The interface takes as input a list of models, taken from a YAML file, and outputs the Evaluation Scores (ROUGE, BLEU and METEOR) for that model.

Listing A.1: app.py

```
iface = gr.Interface(
    fn=evaluate_model,
    inputs=[
```

```
        gr.Dropdown(choices=[model['name'] for model in
            models_config], label="Select Transformer Model"),
    ],
    outputs=[
        gr.JSON(label="Evaluation Scores"),
    ]
```

The `Auto Class` from Hugging Face is used to load the models, as it is able to deduce the model architecture from the name provided. Then, the model is set to evaluation mode - this deactivates certain layers only used during training.

Listing A.2: summarize.py/load_model_and_tokenizer

```
model = AutoModelForSeq2SeqLM
        .from_pretrained(model_name)
        .to(device)
model = model.eval()
```

Inside the summarization function, we define multiple keyword arguments. An important one is `num_beams`, used during Beam Search. Beam search is an optimization algorithm that explores multiple possible sequences (beams) simultaneously to find the most likely output. Also, `no_repeat_ngram_size` is set to 2 in order to avoiding repeating 2-grams and making more abstractive summaries. Finally, if the model supports it, global attention is used. The difference is that, unlike normal attention, which limits each token to attend only to a window of surrounding tokens, the global attention attends to each one in the input, enhancing the understanding of the context.

Listing A.3: summarize.py/summarize_and_score

```
model_kwargs = {
    "input_ids": input_ids,
    "attention_mask": attention_mask,
    "return_dict_in_generate": True,
    "num_beams": 4,
    "no_repeat_ngram_size": 2,
    "early_stopping": True,
    **kwargs
}

if hasattr(model.config, 'attention_window'):
    global_attention_mask = torch.zeros_like(attention_mask)
    global_attention_mask[:, 0] = 1
    model_kwargs['global_attention_mask'] =
        global_attention_mask

summary_pred_ids = model.generate(**model_kwargs)
```

A.3 Fine-Tuning

In order to split the summary into sentences, NLTK Punkt has been used. It is a pre-trained sentence tokenizer that utilizes unsupervised machine learning in order to find sentence boundaries.

For the training step, the Hugging Face Trainer has been used instead of a regular training loop with PyTorch. This is because the Trainer automates various tasks that would otherwise require custom code such as: batching, gradient accumulation and mixed precision training. Also, it integrates with the Hugging Face Hub, making it easy to publish a model. Here is a link to the published `andreiujica/led-base-bigpatent` model. Below is a simplified example of how the Hugging Face Trainer works.

```
training_args = Seq2SeqTrainingArguments(
    learning_rate=4e-05,
    num_train_epochs=6,
    (...)
)

trainer = Seq2SeqTrainer(
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics
)
```