# Automatic Summarization of Long Documents

**Naman Chhibbar**
IIT Hyderabad
Kandi, Sangareddy
Telangana 502285, India
ma21btech11011@iith.ac.in

**Jugal Kalita**
University of Colorado, Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs CO 80918
jkalita@uccs.edu

## Abstract

A vast amount of text is added to the internet daily, making utilization and interpretation of textual data complex and cumbersome. As a result, automatic text summarization is crucial for extracting relevant information, saving precious time. Although many transformer models excel in summarization, they are constrained by their input size, preventing them from processing texts longer than their context size. This study introduces three novel algorithms that allow any large language model to efficiently overcome its input size limitation, effectively utilizing its full potential without any architectural modifications. We test our algorithms on texts with more than 70,000 words, and our experiments show a significant increase in BERTScore with competitive ROUGE scores.

## 1 Introduction

Due to the ever-increasing amount of textual data available online, document summarization has become crucial for the efficient and accurate extraction of relevant information. Large Language Models (LLMs) based on the transformer architecture (Vaswani et al., 2017) have shown outstanding abilities in many NLP tasks, including document summarization (Yadav et al., 2023). Recent developments have demonstrated remarkable improvements in the relevancy and coherence of summaries generated by such LLMs.

However, long document summarization, which involves removing redundancies and makes reading long texts efficient, remains a major challenge. One of the significant limitations in the transformer architecture is limited context size, stemming from the quadratic memory and computational complexity of the attention mechanism (Du et al., 2023). This constraint hinders extracting relevant information from extensive texts where summarization is valuable to overcome the time, effort, and interpretive issues posed by complex and large documents.

We experiment with three novel approaches to address the input size limitations of transformers. The methods introduced do not include any architectural modifications to the model used and can be incorporated into any existing pipeline. We believe that these methods can effectively utilize the full potential of any existing LLM by capturing information from crucial aspects of the document. Though our experiments only include the task of summarization, we believe that our methods can be applied to NLP tasks that require processing long texts.

We start by providing the problem statement (Section 2) and discussing related work (Section 3) to gain insights into the problem and the state-of-the-art solutions. We then introduce the datasets (Section 4) and methodology (Section 5) used in our experiments. For evaluating our results, we present standard metrics (Section 6) used in text summarization. We end the report by discussing our experimental findings (Section 7) and potential future work (Section 8) and concluding the study (Section 9).

## 2 Problem Statement

Our goal is to pre-process and distill a long document (with theoretically indefinite length) such that it fits within the context size of the model while retaining important information. In our experiments, we use documents with lengths of up to **seventeen times** the context size of the model and aim to reduce the summary length to about 400 words or less, preserving maximal salient information and coherence.

## 3 Related Works

There have been efforts to improve the efficiency of the attention mechanisms in transformers. Beltagy et al. (2020) introduce the Longformer, which replaces the quadratic self-attention mechanism in the transformer architecture with a sliding window self-attention, resulting in a linear complexity with respect to the input size. To capture long-range dependencies, they include global attention at specific token positions. Huang et al. (2021) modify the encoder-decoder attention mechanism such that each attention head in the decoder attends to $n/s_h$ tokens in the input sequence, where $n$ is the input length and $s_h$ is the number of heads. This method has a complexity of $O(mn/s_h)$, where $m$ is the length of the output sequence. Bertsch et al. (2023) introduce Unlimiformer, which also modifies the encoder-decoder attention in a transformer. The attention heads in the decoder only attend to the tokens picked by their k-Nearest-Neighbor (kNN) algorithm. The kNN indices between the input tokens are created by the hidden states generated in the encoder. Phang et al. (2022) introduce the staggered block-local attention mechanism. In the block-local attention mechanism, the input sequence is divided into multiple non-overlapping blocks. Tokens in a block attend only to the tokens in the same block. In staggered block-local attention, the blocks are staggered such that each token is in a different block in each head.

Other relevant approaches include VideoAgent, introduced by Wang et al. (2024), an AI agent designed to answer a given question based on a long video. They achieve this by generating captions from multiple uniformly sampled frames from the video. These captions are used to answer the user's question. Chen et al. (2022) describe a novel algorithm to classify long Chinese news into predefined categories. They form multiple groups of sentences based on a maximum token threshold in each group. These groups are then encoded using BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2018) and passed through a 1D convolution layer for local feature extraction. This method is unique because a 1D convolution layer replaces the attention mechanism with linear time complexity. Chen et al. (2023) use positional interpolation to extend the context size of a pre-trained model. Instead of the usual extrapolation of the positional embeddings, they downscale and force them into a range the model is trained on, hence interpolating in the pre-trained range. They claim that the model should use the positional embeddings on which it is trained.

Golia and Kalita (2024) take a "Divide and Conquer" approach to address sequence length limitations in summarizing long meeting transcripts. They begin by segmenting the transcript and then use the BART (Bidirectional and Auto-Regressive Transformer) (Lewis et al., 2020) model to summarize each segment individually. These segment summaries are then recursively combined and summarized until a single summary remains. This method performs well with long documents but may take considerable time to converge due to repeated calls to the model.

## 4 Datasets

This section briefly discusses and analyzes the datasets used in our experiments.

### GovReport

Introduced by Huang et al. (2021), this dataset consists of reports written by government research agencies, including the Congressional Research Service (CRS) and the U.S. Government Accountability Office (GAO). Exact word count information is given in Table 1. Figure 1 shows the word count distribution of the dataset.

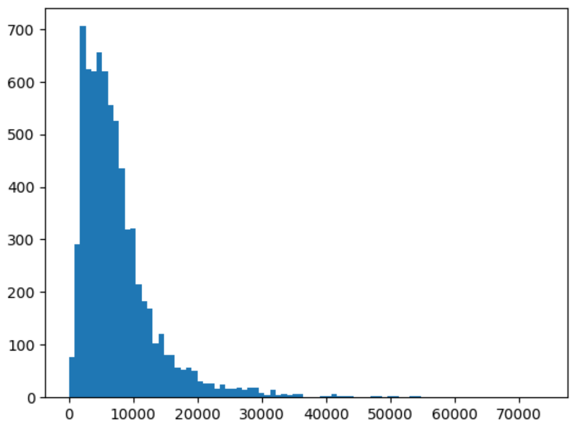

Figure 1: Histogram of GovReport word counts. The x-axis represents document word counts, while the y-axis shows the number of documents.

### BigPatent

Introduced by Sharma et al. (2019), this dataset consists of over 1.3 million records of U.S. patent documents with human-written abstractive summaries. Exact word count information is given in Table 1. Figure 2 shows the word count distribution of the dataset.

## 5 Methodology

In this section, we introduce the three algorithms used in our experiments. Two algorithms start by

| Dataset | Avg. Word Count | Max Word Count | No. of Documents |
|---|---|---|---|
| GovReport | 7,700.71 | **73,815** | 7,238 |
| BigPatent | 3,055.72 | **71,027** | 1,341,362 |

Table 1: Dataset information

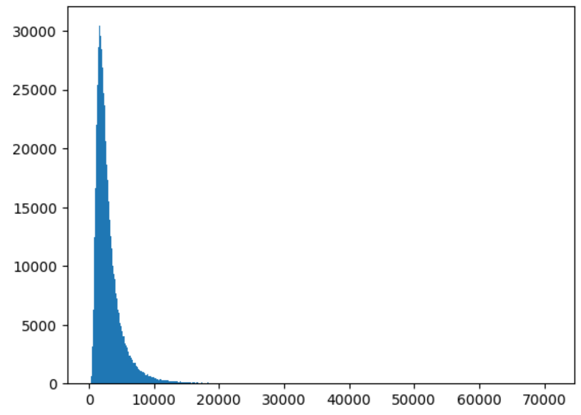

Figure 2: Histogram of BigPatent word counts. The x-axis represents document word counts, while the y-axis shows the number of documents.

segmenting a document into smaller, contiguous, and exhaustive parts. This is done by using a sentence tokenizer to separate sentences in the text and then merging them such that the number of words in each segment is more than the threshold, $min\_words$, a hyperparameter in both methods.

### 5.1 Central Truncation

Truncation is the most common and straightforward approach used to handle long texts that exceed the context size of an LLM. It can be done in three main ways:

- **Retaining Head**: Keeping tokens from the start.
- **Retaining Tail**: Keeping tokens from the end.
- **Head and Tail**: Keeping tokens from both start and end.

Worsham and Kalita (2018) also employ "retaining head" and "retaining tail" strategies on long texts and find promising results for long text genre classification. Though the "retaining head" method is often used, keeping the initial tokens allowed by the LLM, Sun et al. (2019) find that keeping both head and tail produces better results than both the "retaining head" and the "retaining tail" methods. Their research also shows that truncating the middle is even better than the more complicated hierarchical methods, displaying superiority with simplicity. This is a time-efficient method worth exploring.

The fraction of tokens to be taken from the head is controlled by the hyperparameter $head\_size \in [0, 1]$ in our algorithm. Setting $head\_size = 1$ results in taking tokens only from the head, whereas setting $head\_size = 0$ results in taking tokens only from the tail. The truncated tokens are then sent to the model for summarization.

### 5.2 Document Skimming

One way to process long texts is by employing a speed reading strategy called skimming (Dhillon et al., 2020). Skimming is performed by reading the whole text in a go while selectively skipping some parts of the text for quicker reading. The reader usually omits the portions that seem redundant or irrelevant in the text, minimizing information loss. This method is inspired by the way Wang et al. (2024) randomly sample video frames to generate captions. Worsham and Kalita (2018) also use random sampling for genre identification.

This method starts by segmenting the document with the hyperparameter $min\_words$ (introduced at the start of Section 5). We then sample segments uniformly, meaning each segment has an equal probability, $p$, to be picked. The sampled segments are then concatenated to form a single text and are sent to the model. This method ensures the model is exposed to all parts of the text while preserving efficiency. Figure 3 is a visual representation of the algorithm.

Below is an example of the distilled text generated by the algorithm and the summary generated by GPT-3.5 Turbo (Brown et al., 2020):

**Example Text:**

> Title: Awards of Attorneys' Fees by Federal Courts and Federal Agencies. Subsection I. Introduction: The American ...

**Distilled Text:**

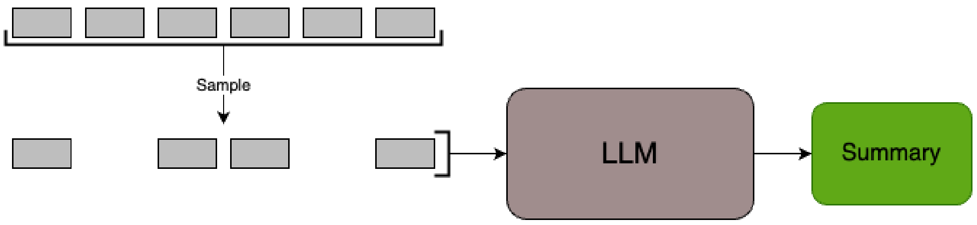

Figure 3: The Document Skimming Algorithm. The grey blocks represent segments of the document.

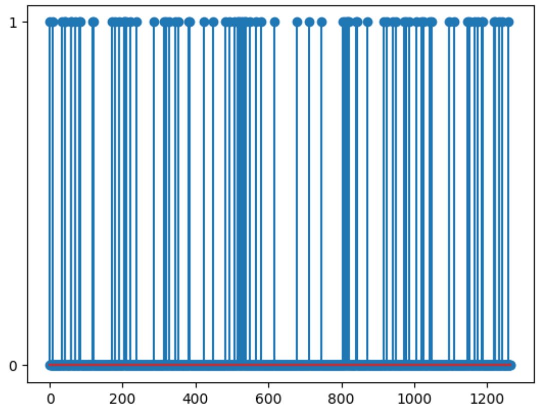

Figure 4: Visualization of the segments picked by the Document Skimming algorithm in a specific long document. The y-axis value of the $i$th segment on the x-axis is one if picked and zero otherwise.

> Alyeska Pipeline Service Co. v. Wilderness Society, 421 U.S. 240, 247 (1975). This is known as the "American rule" (as opposed to the ...

**Summary:**

> The American rule regarding attorneys' fees has two common law exceptions: the common benefit doctrine and bad faith ...

Refer to Figure 4 to visualize the segments the algorithm picks.

### Removing Redundancy

To address the issue of redundancy in the document, we also experiment with removing redundant segments before and after sampling. This is done to prevent the model from seeing the same information multiple times, which may lead to repetition in the output. This is achieved by linearly iterating over the sampled segments and selectively removing some of the segments. We do this by maintaining the mean embedding of the selected segments, initialized as a zero vector. The current segment is retained if the cosine similarity between the mean and segment embeddings is lower than a $threshold$, which is a hyperparameter. A sentence transformer is used to generate the segment embeddings. The sentence transformer is based on MiniLM (Wang et al., 2020), a distilled version of a larger encoder-only transformer model. In case the current segment is retained, the mean embedding is updated as follows:

$$new\_mean = \frac{n \cdot mean\_emb + seg\_emb}{n+1}$$

where $n$ is the number of sampled segments (excluding the current segment), $seg\_emb$ is the segment embedding of the current segment, $mean\_emb$ is the mean embedding, and $new\_mean$ is the updated mean embedding.

While removing segments after sampling, we waste some of the context size. To alleviate this, we oversample the segments beforehand by increasing the probability of choosing a segment. This fraction is controlled by the hyperparameter $prob\_boost$. The updated probability is calculated as follows:

$$p_{new} = (1 + prob\_boost) \cdot p.$$

Even though removing redundant segments before sampling is less efficient due to the whole document being processed, it ensures better utilization of the LLM's context size.

### Other Calculations

We now calculate the optimal probability of picking a segment, $p$. Let $X$ denote the total number of tokens in the sampled segments. Since segments are sampled randomly, $X$ is a random variable. If the context size of the model is $model\_size$, we want $\mathbb{E}[X] = model\_size$, where $\mathbb{E}[X]$ denotes the expectation of $X$.

Suppose we have $n \in \mathbb{N}$ segments and $X_i \sim \text{Bernoulli}(p)$ denotes if segment $i$ is chosen, $i \in \{1, 2, \ldots, n\}$. If $len_i$ denotes the number of tokens in segment $i$, we can write:

$$
\begin{aligned}
X &= \sum_{i=1}^{n} X_i \cdot len_i \\
\Rightarrow \mathbb{E}[X] &= \mathbb{E}\left[\sum_{i=1}^{n} X_i \cdot len_i\right] \\
&= \sum_{i=1}^{n} \mathbb{E}[X_i \cdot len_i] \\
&= \sum_{i=1}^{n} \mathbb{E}[X_i] \cdot len_i
\end{aligned}
$$

Since $X_i \sim \text{Bernoulli}(p)\ \forall i \in \{1, 2, \ldots, n\}$, we have $\mathbb{E}[X_i] = p\ \forall i \in \{1, 2, \ldots, n\}$.

$$
\begin{aligned}
\therefore \mathbb{E}[X] &= \sum_{i=1}^{n} p \cdot len_i \\
&= p \cdot \sum_{i=1}^{n} len_i
\end{aligned}
$$

Let $total\_len$ be the total number of tokens in the text, then $total\_len = \sum_{i=1}^{n} len_i$.

$$
\begin{aligned}
\therefore \mathbb{E}[X] = p \cdot total\_len = model\_size \\
\Rightarrow p \cdot total\_len = model\_size \\
\Rightarrow p = model\_size / total\_len
\end{aligned}
$$

## 5.3 Summarization with Keyword Extraction

Document skimming (subsection 5.2) involves a very intuitive and straightforward approach of sampling segments randomly. To use the entirety of the text, we experiment with an efficient keyword extraction algorithm to get essential keywords that explain the core meaning of the document. These keywords capture the overall meaning of the document and can help us sample segments intelligently, ensuring we get the most important segments from the document.

We use Latent Dirichlet Allocation (LDA) (Blei et al., 2003) with a single topic to get the topic words (keywords) from the document. There are many ways to use these keywords; a simple one we use is to concatenate the keywords using a delimiter (a space is used in our experiments) to form a single sentence. This sentence is then embedded to form the keyword embedding, which, in theory, captures a high-level meaning of the document. The keyword sentence and document segments are embedded using the same sentence transformer used in the previous method. The segment embeddings are then compared to the keyword embedding using cosine similarity to get similarity scores for each segment embedding. The maximum possible number of segments with the highest similarity scores are retained, concatenated and sent to the model for summarization. Algorithm 1 describes the process.

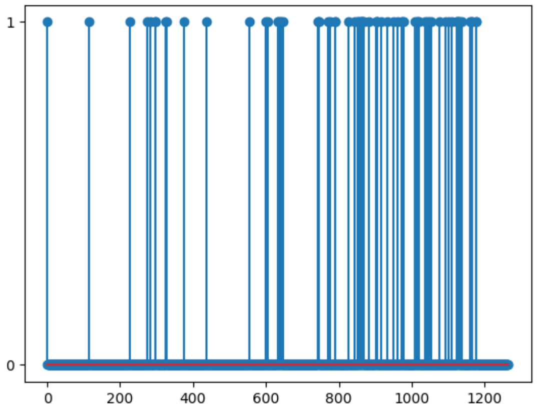

Figure 5: Visualization of the segments picked by the Summarization with Keyword Extraction algorithm in a specific long document. The y-axis value of the $i$th segment on the x-axis is one if picked and zero otherwise.

Below is an example of the distilled text generated by the algorithm and the summary generated by GPT-3.5 Turbo (Brown et al., 2020):

**Example Text:**

> Title: Awards of Attorneys' Fees by Federal Courts and Federal Agencies. Subsection I. Introduction: The American ...

**Distilled Text:**

> Title: Awards of Attorneys' Fees by Federal Courts and Federal Agencies Subsection I. Introduction: The American Rule and ...

**Summary:**

> The document discusses the American Rule regarding attorneys' fees, where prevailing litigants are not typically entitled to ...

Refer to Figure 5 to visualize the segments the algorithm picks.

This approach is similar to how Golia and Kalita (2024) use action items to pick segments of text (a neighbourhood of 2 sentences around the action item) to obtain meeting minutes.

**Algorithm 1** Summarization with Keyword Extraction

```
Input: $text$ (text), $size$ (context size of model)
Output: Distilled text
$segments \leftarrow$ segmenter($text$)
$embeddings \leftarrow$ sentence_transformer($segments$)
$keywords \leftarrow$ LDA($text$)
concatenate($keywords$, delimiter)
$keyword\_embedding \leftarrow$ sentence_transformer($keywords$)
Sort $embeddings$ by decreasing cosine similarity scores with $keyword\_embedding$
$selected \leftarrow \{\}$
$num\_tokens \leftarrow 0$
for $embedding \in embeddings$ do
    $tokens \leftarrow$ count_tokens($embedding$)
    if $tokens + num\_tokens \leq size$ then
        $selected \leftarrow selected \cup \{embedding\}$
        $num\_tokens += tokens$
    end if
end for
concatenate($selected$, delimiter)
return $selected$
```

## 6 Evaluation Metrics

The best way to evaluate generated natural language is by humans, but conducting human trials is expensive and time-consuming. Due to this difficulty, we use automatic evaluation metrics to evaluate the quality of the generated summary given some reference summaries. Fabbri et al. (2021) review many such open-source and state-of-the-art metrics. The two that we use in our experiments are discussed below. These metrics are commonly used in published literature.

**ROUGE metrics:** Lin (2004) introduces the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) metrics. The basic ROUGE-N metric is based on the fraction of overlaps of ideal or reference summaries with the candidate summary, hence being recall-oriented. His study concludes that ROUGE-N with N $= 2$, ROUGE-L, ROUGE-W, and ROUGE-S work well for the summarization task.

**BERTScore:** Zhang et al. (2019) introduce BERTScore, an automatic evaluation metric for text generation. BERTScore is calculated by comparing the contextual embeddings of tokens in the candidate and reference summaries, which are generated using BERT (Devlin et al., 2018). BERTScore excels at capturing semantic similarities between sentences since it uses contextual embeddings of tokens instead of N-gram frequencies to calculate similarity.

## 7 Experimental Findings

We test our pipelines with the following models: **BART** fine-tuned on the CNN/Daily Mail dataset (Nallapati et al., 2016) with a context size of 1024, **LongT5** (Guo et al., 2021), a variant of T5 (Text-to-Text Transfer Transformer) (Raffel et al., 2020), fine-tuned on the BookSum dataset with a context size of 4096, and **GPT-3.5 Turbo** (Brown et al., 2020) with a context size of 4096.

We compare our results with the state-of-the-art summarization models on the GovReport dataset, including Unlimiformer (Bertsch et al., 2023) integrated with BART and PRIMERA (Beltagy et al., 2020), Hepos (Huang et al., 2021), PEGASUS-X with staggered block-local attention (Phang et al., 2022), extended LLaMA-7B with positional interpolation (Chen et al., 2023). We also compare our results with BigBird-Pegasus (Zaheer et al., 2020) on the BigPatent dataset. Refer to Table 2 and Table 3 for results on the GovReport and BigPatent datasets, respectively.

We could not obtain the BERTScores of our baselines, except for Unlimiformer, due to the unavailability of code or computational limitations.

### Time complexity analysis

We evaluate the time complexity of our methods by measuring the mean time taken to process a

| Model | ROUGE-1 | ROUGE-2 | ROUGE-L | BERTScore |
|---|---|---|---|---|
| BART w/ Unlimiformer (1,024) | 53.4 | 22.5 | 22.5 | 66.0 |
| PRIMERA w/ Unlimiformer (4,096) | 56.5 | 24.8 | 26.3 | 67.7 |
| Hepos (10,240) | 51.34 | 19.09 | **48.73** | - |
| PEGASUS-X w/ Staggered Block-Local Attention (16k) | 60.3 | **30.0** | 31.5 | - |
| LLaMA-7B w/ Positional Interpolation (15k) | 60.0 | 28.0 | 29.5 | - |
| Summarization w/ Extraction + GPT-3.5 Turbo (4,096) | **61.99** | 18.52 | 38.46 | **86.20** |
| Central truncation + LongT5 (4,096) | 46.20 | 4.38 | 38.27 | **82.19** |
| Skimming w/ post-sampling removal + LongT5 (4,096) | 46.76 | 4.56 | 39.61 | **81.96** |

Table 2: Automatic evaluation results on the GovReport dataset. The context sizes of the models are mentioned in parentheses. The best score in each metric category is highlighted in **bold**. The results of our algorithms are below the horizontal line in the middle.

| Model | ROUGE-1 | ROUGE-2 | ROUGE-L | BERTScore |
|---|---|---|---|---|
| BigBird-Pegasus (16k) | **60.64** | **42.46** | **50.01** | - |
| Skimming w/ pre-sampling removal + GPT-3.5 Turbo (4,096) | 27.40 | 3.31 | 21.25 | **82.62** |
| Central truncation + GPT-3.5 Turbo (4,096) | 27.77 | 3.09 | 20.56 | **82.57** |
| Skimming w/ post-sampling removal + GPT-3.5 Turbo (4,096) | 26.16 | 2.13 | 20.21 | **82.40** |

Table 3: Automatic evaluation results on the BigPatent dataset. The context sizes of the models are mentioned in parentheses. The best score in each metric category is highlighted in **bold**. The results of our algorithms are below the horizontal line in the middle.

document (excluding the time taken by the model to generate the summaries). We find that "Central Truncation" (subsection 5.1) and "Document Skimming" (subsection 5.2) take approximately the same time. "Skimming with post-sampling removal" (removing segments after sampling) takes slightly longer than the other two methods. We can see a significant increase in time taken by "Skimming with pre-sampling removal" (removing segments after sampling) and "Summarization with Keyword Extraction" (subsection 5.3) due to the additional computations required. Figure 6 illustrates the average time taken by our methods. Check Table 4 for exact values rounded off to two decimal places.

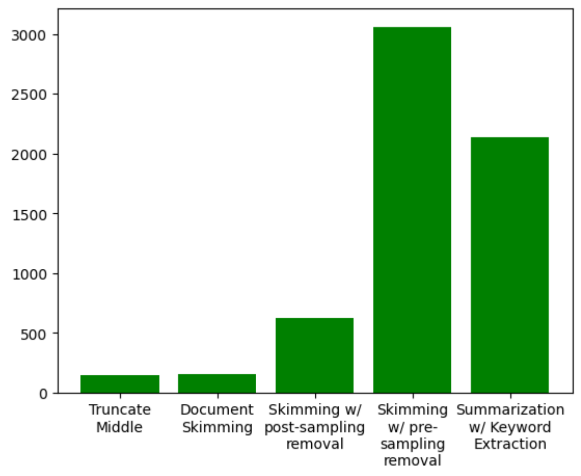

Figure 6: Mean time taken (in milliseconds) per document using BART tokenizer on BigPatent dataset

## 8 Future Work

To segment the document, we use a basic sentence tokenizer (nltk.sent_tokenize) with some modifications to control the minimum number of words in a segment. In our experiments, we find that segmentation is a crucial step in the distillation process and can greatly influence the output summary, indicating that good segmentation is vital for good distillation of text. Ensuring the uniformity of the length of the segments while preserving coherence within a segment is also essential for better utiliza-

| Method | Mean time taken |
|---|---|
| Central Truncation | 142.50 ms |
| Document Skimming | 155.42 ms |
| Skimming w/ post-sampling removal | 625.17 ms |
| Skimming w/ pre-sampling removal | 3059.63 ms |
| Summarization w/ Extraction | 2131.40 ms |

Table 4: Mean time taken (in milliseconds) per document using BART tokenizer on BigPatent dataset

tion of the context size of the model. We encourage future work to experiment with different kinds of segmenters.

Future work can also focus on extending the "Summarization with Keyword Extraction" method (subsection 5.3). Many potential ways exist to use the extracted keywords we do not touch upon.

## 9 Conclusion

Our experiments show that "Document Skimming with post-sampling removal" (subsection 5.2) performs well while being efficient. The "Central Truncation" method (subsection 5.1) also shows good results, which shows that simple methods can also be effective when dealing with long inputs. The last two methods, "Skimming with pre-sampling removal" (subsection 5.2) and "Summarization with Keyword Extraction" (subsection 5.3), achieve the best results but are computationally expensive.

Our experiments show significant improvement in BERTScore compared to Unlimiformer (Bertsch et al., 2023) on the GovReport dataset, showing that our pipelines can efficiently utilize details in long texts. Even though our ROUGE-2 scores are lower than the baselines, ROUGE-1 and ROUGE-L scores are competitive. Since BERTScore is better at capturing semantic similarity, we highlight the use of BERTScore compared to ROUGE scores. Hence, we contend that our pipelines can generate better summaries than the baselines with higher ROUGE scores. Also, note that the models used in our experiments have smaller context sizes than the baselines, indicating that our algorithms have a greater potential if used with larger models.

## Acknowledgement

All work herein reported is supported by the Nation Science Foundation under Grant No. 2349452. Any opinion, finding, or conclusion in this study is that of the authors and does not necessarily reflect the views of the National Science Foundation.

## Supplementary Materials

The datasets used in this study are available here: GovReport, BigPatent

The code used in this study is available here: GitHub

## References

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew Gormley. 2023. Unlimiformer: Long-range transformers with unlimited length input. In *Advances in Neural Information Processing Systems*, volume 36, pages 35522–35543. Curran Associates, Inc.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*.

Xinying Chen, Peimin Cong, and Shuo Lv. 2022. A long-text classification method of chinese news based on bert and cnn. *IEEE Access*, 10:34046–34057.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Bobby Pramjit Singh Dhillon, Herman Herman, and Syafryadin Syafryadin. 2020. The effect of skimming method to improve students'ability in reading comprehension on narrative text. *Linguists: Journal Of Linguistics and Language Teaching*, 6(1):77–88.

Jiangsu Du, Jiazhi Jiang, Jiang Zheng, Hongbin Zhang, Dan Huang, and Yutong Lu. 2023. Improving computation and memory efficiency for real-world transformer inference on gpus. *ACM Trans. Archit. Code Optim.*, 20(4).

Alexander R Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. 2021. Summeval: Re-evaluating summarization evaluation. *Transactions of the Association for Computational Linguistics*, 9:391–409.

Logan Golia and Jugal Kalita. 2024. Action-item-driven summarization of long meeting transcripts. In *Proceedings of the 2023 7th International Conference on Natural Language Processing and Information Retrieval*, NLPIR '23, page 91–98, New York, NY, USA. Association for Computing Machinery.

Mandy Guo, Joshua Ainslie, David Uthus, Santiago Ontanon, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. 2021. Longt5: Efficient text-to-text transformer for long sequences. *arXiv preprint arXiv:2112.07916*.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

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

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

Jason Phang, Yao Zhao, and Peter J Liu. 2022. Investigating efficiently extending transformers for long input summarization. *arXiv preprint arXiv:2208.04347*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Eva Sharma, Chen Li, and Lu Wang. 2019. BIGPATENT: A large-scale dataset for abstractive and coherent summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2204–2213, Florence, Italy. Association for Computational Linguistics.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? In *Chinese computational linguistics: 18th China national conference, CCL 2019, Kunming, China, October 18–20, 2019, proceedings 18*, pages 194–206. Springer.

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

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788.

Xiaohan Wang, Yuhui Zhang, Orr Zohar, and Serena Yeung-Levy. 2024. Videoagent: Long-form video understanding with large language model as agent. *arXiv preprint arXiv:2403.10517*.

Joseph Worsham and Jugal Kalita. 2018. Genre identification and the compositional effect of genre in literature. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1963–1973, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Avaneesh Kumar Yadav, Ranvijay, Rama Shankar Yadav, and Ashish Kumar Maurya. 2023. State-of-the-art approach to extractive text summarization: a comprehensive review. *Multimedia Tools and Applications*, 82(19):29135–29197.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.