Automated Judicial Case Briefing

Yoobin Cheong
New York University
yc5206@nyu.edu

Yeong Koh
New York University
yk2678@nyu.edu

Yoon Tae Park
New York University
yp2201@nyu.edu

Abstract

Our project aims to build a model that extracts five distinct sections from a judicial case opinion using natural language processing techniques such as abstractive summarization methodologies. While this project can be done by using current SOTA summarization transformers, we've faced some critical challenges such as having extremely long input, and needs to summarize 5 different parts or contents of the input. As a result of various approaches with experiments, we have successfully created a pipeline that uses SOTA models (i.e. BART) that shows a score of 0.3 for the baseline and 0.47 for the fine-tuned model. A model that can generate acceptable outputs will potentially be applied in the legal domain of the World Bank after the law practitioners validate the quality of the model outputs. Our experiments have shown that using larger models and text extraction methods did not improve the performance of the model, but other approaches could be explored in order to improve its performance. Further research and experimentation will be necessary in order to develop a model that can accurately and effectively summarize complex legal documents.

1 Introduction

A case brief is a dissection of a judicial opinion – it contains a written summary of the basic components of that decision. From the opinions, we extract five distinct sections from a case brief, a dissection of a judicial opinion that contains a written summary of the basic components of those decisions. The five tasks that we generate with our model are the “Justia summary", “Justia holding", “Facts of the case", “Question", and “Conclusion" (Figure 1).

This text summarization task, or the task of producing a concise and fluent summary while preserving key information content and overall meaning (Allahyari et al. [1]), is very human intensive in nature as it requires a lot of time for humans to read through long case texts and to write a summary for each section that captures relevant information. Expedition of the process can save a lot of human effort and also help law practitioners refer to multiple case documents at ease and speed.

The goal of our project is to develop a natural language processing model that uses summarization techniques, such as abstractive summarization, to extract five distinct sections from a judicial case opinion. We create a pipeline that uses state-of-the-art models like BART (Lewis et al. [7]). We use the Rouge-L score (Lin [8]) to measure the performance of our models, which is a metric based on the longest common subsequence (LCS) that is shared between the model output and golden label. A longer shared sequence indicates a higher level of similarity between the two sequences.

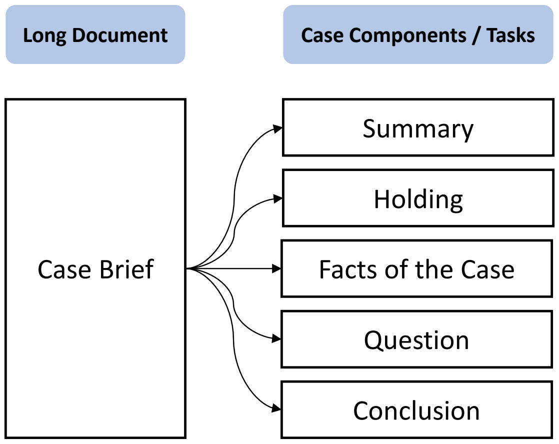


Figure 1: Input (Case Brief) and Outputs (5 Tasks)

2 Related Work

In general, text summarizations can be divided into two main types: extractive summarization and abstractive summarization.

Extractive summarization is a method of creating a summary by selecting a few key sentences or phrases from the original text and using them to form the summary. As a result, the sentences or phrases in an extractive summary are all taken directly from the original text. One well-known algorithm for extractive summarization is TextRank (Mihalcea and Tarau [9]). TextRank is a technique for summarizing documents by modifying Google's PageRank algorithm. PageRank represents documents as a graph structure and continuously updates the importance of the nodes to generate rankings. However, a disadvantage of this approach is that the model's language generation ability is limited because it can only use existing sentences and phrases.

To improve on this, abstractive summarization is a method of summarizing the original text by generating new sentences or phrases that reflect the main context of the original text, even if they were not present in the original text. Abstractive summarization is like a human summarizing a text, where they can use their own words and sentences to convey the main points of the original text. Abstractive summarization algorithms often rely on seq2seq models, and different algorithms can be used in combination to produce the most accurate summaries. For example, Seq2seq models with attention mechanisms (Chopra et al. [4]) and pointer-generator methods (See et al. [13]) allow the model to focus on specific parts of the input text, enabling it to generate summaries that accurately reflect the content of the original text.

Extractive summarization and abstractive summarization can be performed in more advanced ways using BERT (Devlin et al. [5]), GPT-2(Radford et al. [11]), BART (Lewis et al. [7]), or T5 (Raffel et al. [12]), respectively. BERT can be used for extractive summarization by identifying important sentences in the original document, and GPT-2, BART, or T5 can be used for abstractive summarization by generating a summary of the document in a new, coherent form. Although these methods have shown good performance on summarization datasets that contain small documents, they may not be as effective on longer documents or in low-resource environments (Bajaj et al. [2]).

3 Problem Definition and Methodology

3.1 Task

There are many examples of long documents, but one type that is particularly well-suited for addressing the challenge of long document summarization is legal documents. For example, after each trial, the Supreme Court organizes the records and documents and creates a summary of the case. This summary includes the specific facts and history that influenced the outcome of the trial, as well as any legal controversies that arose during the trial. This process of reading and organizing the content of a trial is labor-intensive and time-consuming.

To automate this process, there exists many text summarization models using current SOTA transformers. The major challenge in most summarization models is that they have a limitation in length of input tokens, so they are not suitable for legal documents that have significantly longer text. For example, the BART model can only have maximum input of 1,024 tokens while some legal documents can have more than 0.3 million words. Therefore, while these methods work well for short documents, it is still difficult to expect a certain quality level of summarization for long legal documents where the model cannot use the entire document as an input.

In this project, our primary task is to build a model that automates the case summarization process and extracts relevant information from extremely long legal documents. Using the Supreme Court's public case opinion texts as an input, our model is expected to output five different summarizations. Below table shows a sample case from the actual case opinions (Table 1).

Input (truncated) NOTE:Where it is feasible, a syllabus (headnote) will be released, as is being done in connection with this case, at the time the opinion is issued. The syllabus constitutes no part of the opinion of the Court but has been prepared by the Reporter of Decisions for the convenience of the reader. See United States v. Detroit Timber & Lumber Co.,200 U.S. 321, 337. SUPREME COURT OF THE UNITED STATES Syllabus Oklahoma v. Castro-Huerta certiorari to the court of criminal appeals of oklahoma No. 21429. Argued April 27, 2022 Decided June 29, 2022 In 2015, respondent Victor Manuel Castro-Huerta was charged by the State of Oklahoma for child neglect. Castro-Huerta was convicted in state court and sentenced to 35 years of imprisonment. While Castro-Huertas state-court appeal was pending, this Court decided ...
Output - Summary (truncated) Castro-Huerta was convicted of child neglect in Oklahoma state court. The Supreme Court subsequently held that the Creek Nation's eastern Oklahoma reservation was never properly disestablished and remained "Indian country." Castro-Huerta then argued that the federal government had exclusive jurisdiction to prosecute him (a non-Indian) for a crime committed against his stepdaughter (Cherokee Indian) in Tulsa (Indian country). The Oklahoma Court of Criminal Appeals vacated his ...
Output - Holdings The federal government and the state have concurrent jurisdiction to prosecute crimes committed by non-Indians against Indians in Indian country.
Output - Facts Victor Manuel Castro-Huerta, a non-Native, was convicted in Oklahoma state court of child neglect, and he was sentenced to 35 years. The victim, his stepdaughter, is Native American, and the crime was committed within the Cherokee Reservation.Castro-Huerta challenged his conviction, arguing that under the Supreme Court's 2020 decision in McGirt v. Oklahoma, which held that states cannot prosecute crimes committed on Native American lands without federal approval. Oklahoma argued that McGirt involved a Native defendant, whereas Castro-Huerta is non-Native, so McGirt does not bar his prosecution by the state.
Output - Question Do states have the authority to prosecute non-Natives who commit crimes against Natives on Native American lands?
Output - Conclusion (truncated) The federal government and the state have concurrent jurisdiction to prosecutecrimes committed by non-Natives against Natives on Native American land. Justice Brett Kavanaugh authored the majority opinion of the Court.The Court has held that States have jurisdiction to prosecute crimes committed by non-Natives against non-Natives on Native American lands. Native American land is not separate from state territory. And States have jurisdiction to prosecute crimes committed on ...

Table 1: Input and Output of Sample Case

1) Summary: Summarization of the entire case opinion.
2) Holding: Court's decision of the case.
3) Facts of the case: Detail information of the event that are legally relevant to the court's decision including history of the dispute, legal claims, and defenses.
4) Question: Statement of the question of law that the court must answer in order to make a decision.
5) Conclusion: Decision made by a judge regarding a question of law.

To improve our model, we fine-tune the baseline model and apply different methods. We evaluate the performance using the Rouge-L score, which compares the longest common sub-sequences. While the Rouge-L score can provide an estimate of accuracy based on the number of common words, it may not accurately reflect the quality of the summary. Therefore, we use the Rouge-L score only as a reference for fine-tuning and not as an exact measure of the quality of summarization.

3.2 Data

A dataset of 8,236 case opinion texts that has been webscraped from Justia US Supreme Court website (`https://supreme.justia.com/`) is used for our project. The cases in concern dates back from 1966 to 2021 where each row of the dataset has information about each of the cases including the name, the docket number, term (or year), and a short description of the case. The columns we are concerned with are the columns that are labeled as the following: “Justia summary", “Justia holding", “Facts of the case", “Question", and “Conclusion". We define the five distinctive summarizations for each of these components as a task. These columns, after proper preprocessing of the texts, will be used to fine-tune each of our transformer models.

As a part of data processing, we inspect our dataset for any anomalies. We first remove HTML tags from the documents to extract clean tokens. Then, we look at the length of the input documents and the length of each task. We generate a distribution of the length of tokens for each task in order to identify any outliers, such as unusually short or long tokens. For example, we find two cases with token length 2 that contained the phrases “Currently available" and “Currently unknown". These rows are interpreted as having no useful information, so we drop them from the dataset. Additionally, we remove any rows that are null from the training set prior to fine-tuning our model for each task.

For each task, there are varying proportions of null values in our training set and the rows containing nulls are all dropped from our dataset prior to fine-tuning our model for each task. After removal of null or insufficient cases, we are left with the following numbers of document-summary pairs; 1,278 pairs for “Justia summary", 781 pairs for “Justia holding", 3,357 pairs for “Facts of the case", 3,356 pairs for “Question", and 3,356 pairs for “Conclusion".

The input document of our model has an average token length of 14,594, with length that ranges from 95 to 158,087, which is significantly long compared to those of five different summaries. The main challenge of our project is to design a model that can handle these varying input document lengths and generate a unique summary for each task by identifying relevant information. A table showing the mean, minimum, and maximum length of the outputs is shown below (Table 2).

	Mean	Min	Max
Total Input	14,594	95	158,087
Summary Input	13,090	148	83,690
Summary Output	351	60	692
Holdings Input	16,438	148	139,484
Holdings Output	40	10	224
Facts Input	14,478	59	189,086
Facts Output	210	14	1,203
Questions Input	14,485	59	189,086
Questions Output	37	6	198
Conclusion Input	14,479	59	189,086
Conclusion Output	192	9	1,032

Table 2: Mean, Min, Max Token Length of Input and Output

3.3 Methodology

3.3.1 Experiment approaches

This section outlines our experimental approaches (Figure 2).

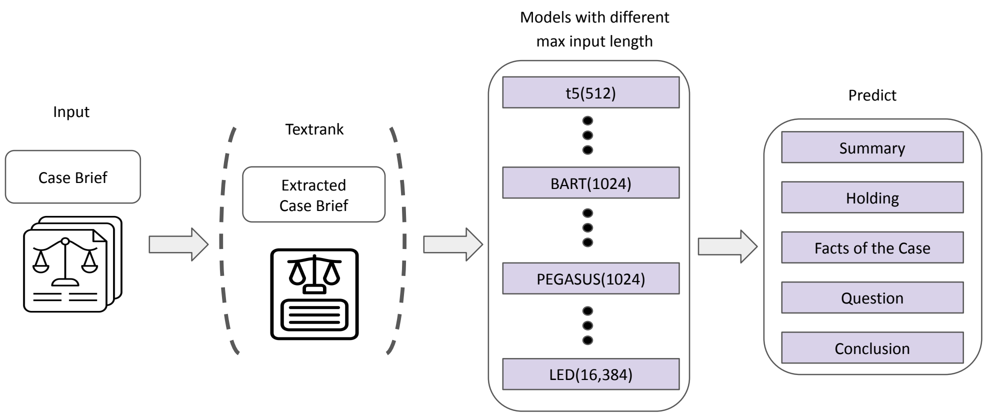


Figure 2: Experimental Design

Approach 1 Building a pipeline that uses current SOTA transformer model, BART
We first create a pipeline that takes case opinion text as input and outputs a distinctive summary for each of the five tasks. We use several SOTA transformer models that are well-known for their performance on summarization tasks. Specifically, we experiment with BART, T5, and PEGASUS (Zhang et al. [14]).

For evaluating the performance of our models, we use the Rouge-L score as a metric for comparison. Rouge-L is based on the longest common subsequence (LCS) shared between the model output and the reference. A longer shared sequence indicates a higher level of similarity between the two sequences. We use an 80-20 split on our dataset and experiment with 10 different random seeds for each model to assess the robustness of the model by checking for consistency in the Rouge-L score.

Baseline Rouge-L scores of our models are used as our starting point and we apply different methods found in other related works in order to track any improvements made to our models. In other words, using Rouge-L as our metric, we experiment with various methods to see if they help us achieve higher scores for any of our tasks.

Approach 2 Using a transformer model (LED) that handles longer documents
The main summarization task models that we mentioned under Step 1 have a limitation on maximum input token lengths. For example, Bart can only take in as its input maximum of 1,024 tokens and truncates the rest of the documents. This means that our model only looks at only 1/10 of the entire document length and generate a summary from only the beginning of the document. Therefore, we applied an LED (Longformer-Encoder-Decoder, Beltagy et al. [3]), which is designed for longer documents, with the capacity to handle at most 16,384 tokens.

Approach 3 Extracting relevant sentences
For documents that exceed the model's capacity, we extract relevant sentences or remove irrelevant sentences by using sentence ranking methodology.
Even with alternative models that can deal with longer documents, there still exist some documents that are longer than the model's capacity. We have tried the TextRank method that can extract and remove irrelevant sentences to compress documents down to the model's capacity limit.

Approach 4 Using LED followed by sentence extraction
We combine Approach 2 and Approach 3 to apply a sentence extracting strategy on an LED model.

3.3.2 Experiment structure

We have defined our experimental structure as below:

Dataset As explained previously, we use cleaned version of 5 different task input-output pairs for the baseline experiment. For additional experiments, we apply some text extraction methods. For these approaches, input-output pairs are updated based on each algorithm. Detailed explanation of the algorithms are explained in **3.3.4 Algorithm**.

Models/architectures We apply various encoder-decoder transformer architectures. As a baseline approach, we use T5, BART, PEGASUS pre-trained models, as those are widely used for summarization tasks. Additionally, we apply LED model as it is capable of handling at most 16,384 input tokens. We use LED model by changing its maximum input tokens from 1,024 to 16,384 and compare the performance of the models.

Experimental setup First, we split the train and test set with an 8:2 ratio using a random seed of 0. Then, we define the maximum input token length based on the maximum input length of the model. However, for the LED model, we change the input token length from 1,024 to 16,384 in order to compare the performance of LED with the other models when the input token length is set to the same value. The output length is fixed to 512, as the longest token length in the output is less than 512. We use a batch size of 2, a learning rate of 5e-05, weight decay of 0.02, and 10 epochs as our defualt hyperparmeter setting. We also set an early stopping point based on the Rouge-L score. We encode the input and output, train the model, and generate predictions for the summary on the test set. We decode the prediction and evaluate the result based on Rouge-L score. We expand this architecture to 5 different tasks.

3.3.3 Evaluation

For each task, we evaluate the performance of the model using the Rouge-L score metric. This metric computes the longest common subsequence(LCS) between two pieces of text (Lin [8]).

$$ROUGE-L(pred, actual) = \frac{(1+B^2)R_{lcs}(pred, actual)P_{lcs}(pred, actual)}{R_{lcs}(pred, actual) + B^2 P_{lcs}(pred, actual)} \tag{1}$$

where

$$P_{lcs} = \frac{LCS_{\cup}(X,Y)}{m},\ R_{lcs} = \frac{LCS_{\cup}(X,Y)}{n},\ m = len(X),\ n = len(Y) \tag{2}$$

$$LCS_{\cup}(X,Y) = \bigcup_{r_i \in Y} \{w | w \in LCS(X, r_i)\} \tag{3}$$

3.3.4 Algorithm

There are several algorithms that we experimented with to improve our baseline results. First, we tried using extraction methods before applying abstraction, by referencing extract & abstract methods conducted by other papers. (Bajaj et al. [2], Elaraby and Litman [6], Moro and Ragazzi [10]) Also, as for efficiency, we tried merging the outputs of 5 tasks and regard the combined output as a single task that generates 5 different sections under a single summary prediction output.

Text Ranking Text ranking is one of the popular methods, which focuses on the extraction of important sentences from a document. This method uses a graph-based ranking model that compares the similarity between sentence vectors. In our project, we apply this method to extract the most important sentences from the original document based on the maximum input token length of our summarization model. Detailed steps are as follows:

Algorithm 1 Text Ranking

```
 1: Use PunktSentenceTokenizer to split the document into sentences
 2: for $each\ sentences = 1, 2, \ldots$ do
 3:     Find vector representation by using CountVectorizer
 4:     Normalize vectors by using TfidfTransformer
 5: end for
 6: Convert the matrix into a graph, and rank sentences based on Pagerank
 7: Create a new empty string $full\ sentence$
 8: while $full\ sentence < maximum\ input\ token\ length$ do
 9:     Append top ranked sentence to the full sentence
10: end while
11: return $full\ sentence$
```

1. Use PunktSentenceTokenizer (`https://www.nltk.org/_modules/nltk/tokenize/punkt.html`) to split document into sentences,
2. For each sentences in the document, find vector representation by using CountVectorizer (`https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer`), and normalize by TfidfTransformer (`https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html`).
3. Create a graph by multipling normalized vectors and its transposed vectors.
4. Use pagerank algoritm to rank each sentences.
5. Sort sentences by its ranking.
6. Create an empty string sentence and append top sentences until it reaches the maximum input token length.
7. Return newly generated full sentence.

Text Segmentation Text segmentation is another extraction methods different from text ranking method, as it is a method of splitting a document into smaller chunks, with each chunk containing at most the maximum input token length for the summarization model. Our project uses the semantic self-segmentation method for text segmentation. This approach uses semantic analysis to identify the most important concepts in the document and then splits the text into segments based on these concepts. This allows the model to accurately summarize the document without losing important information (Moro and Ragazzi [10]).

Algorithm 2 Text Segmentation

```
 1: Create $Final\ chunk$ = [], $current\ chunk$ = [], $next\ chunk$ = []
 2: for $each\ input\ sentences = 1, 2, \ldots$ do
 3:     Calculate cosine similarity on both chunks, and append the sentence
 4:     if chunk reaches the max token length then
 5:         Save the full chunk into $Final\ chunk$ and create a new chunk
 6:     end if
 7: end for
 8: Create $target\ chunk$ that contains empty lists by the number of input chunks
 9: for $each\ target\ sentences = 1, 2, \ldots$ do
10:     Calculate Rouge-L score for all input chunks
11:     Find the higest score of the index and append to the index of $target\ chunks$
12: end for
13: return ($Final\ chunk$, $target\ chunk$) pairs
```

1. Create an empty chunk and next chunk that can have at most maximum input token length.
2. For each input sentences, calculate cosine similiarity on current chunk and next chunk.
3. Compare cosine similarity and append the sentence to the chunk that has higher similarity.
4. Iterate step 1 to step 4 until every sentences are placed into each chunks.
5. Create empty target chunks by number of input chunks.
6. For each target sentences, calculate rogue score for all input chunks.

7. Find the highest score of the index and append to the index of target chunk.
8. Finally, return (chunk, target) pairs.

Task Merge Task merge is a method we use to improve the efficiency of conducting the multiple summarization tasks. By merging the outputs from the five tasks into five separate paragraphs, we generate one input-output pair for each document. This allows us to obtain five different summarization results at once. Detailed steps are as follows:

Algorithm 3 Task Merge

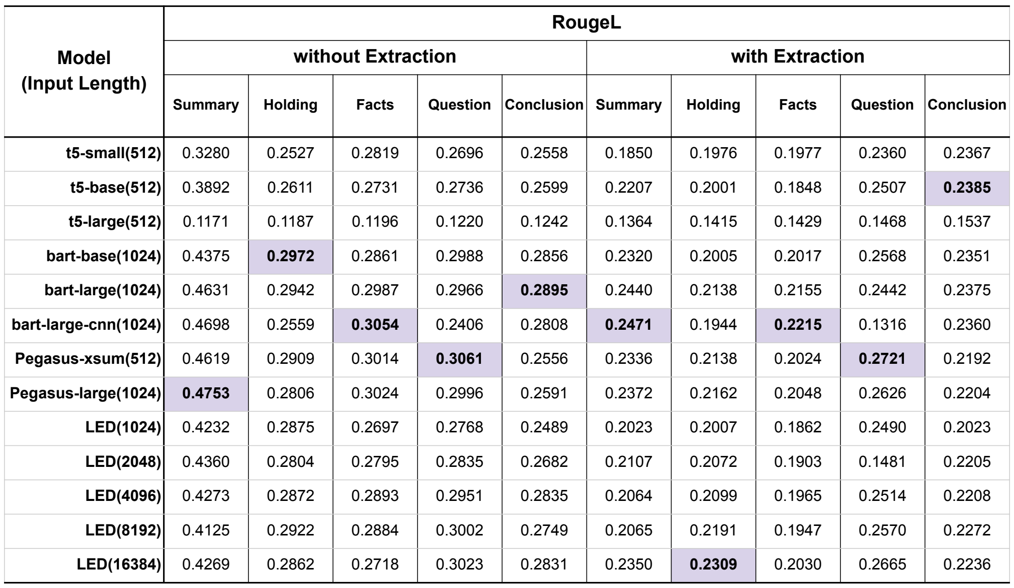
```
1: for $each\ document = 1, 2, \dots$ do
2:     Merge 5 different task outputs by adding task name as a prefix
3: end for
4: for $each\ document = 1, 2, \dots$ do
5:     Run Summarization model and generate comprehensive prediction
6:     Divide prediction by task name prefix
7:     Assign each predictions to actual task summaries and calculate RougeL score
8: end for
9: return Rouge-L score for 5 different tasks
```

1. For each document, merge 5 different outputs by adding task name as a prefix.
2. Run summarization model and get the prediction.
3. Divide predictions by task name.
4. Assign each predictions to actual summary and calculate RogueL score.
5. Return Rogue-L scores for 5 different tasks at once.

4 Experimental Evaluation

4.1 Results

The table below shows the Rouge-L scores based on different models and approached that were defined previously (Figure 3).



Model (Input Length)	RougeL									
	without Extraction					with Extraction				
	Summary	Holding	Facts	Question	Conclusion	Summary	Holding	Facts	Question	Conclusion
t5-small(512)	0.3280	0.2527	0.2819	0.2696	0.2558	0.1850	0.1976	0.1977	0.2360	0.2367
t5-base(512)	0.3892	0.2611	0.2731	0.2736	0.2599	0.2207	0.2001	0.1848	0.2507	**0.2385**
t5-large(512)	0.1171	0.1187	0.1196	0.1220	0.1242	0.1364	0.1415	0.1429	0.1468	0.1537
bart-base(1024)	0.4375	**0.2972**	0.2861	0.2988	0.2856	0.2320	0.2005	0.2017	0.2568	0.2351
bart-large(1024)	0.4631	0.2942	0.2987	0.2966	**0.2895**	0.2440	0.2138	0.2155	0.2442	0.2375
bart-large-cnn(1024)	0.4698	0.2559	**0.3054**	0.2406	0.2808	**0.2471**	0.1944	**0.2215**	0.1316	0.2360
Pegasus-xsum(512)	0.4619	0.2909	0.3014	**0.3061**	0.2556	0.2336	0.2138	0.2024	**0.2721**	0.2192
Pegasus-large(1024)	**0.4753**	0.2806	0.3024	0.2996	0.2591	0.2372	0.2162	0.2048	0.2626	0.2204
LED(1024)	0.4232	0.2875	0.2697	0.2768	0.2489	0.2023	0.2007	0.1862	0.2490	0.2023
LED(2048)	0.4360	0.2804	0.2795	0.2835	0.2682	0.2107	0.2072	0.1903	0.1481	0.2205
LED(4096)	0.4273	0.2872	0.2893	0.2951	0.2835	0.2064	0.2099	0.1965	0.2514	0.2208
LED(8192)	0.4125	0.2922	0.2884	0.3002	0.2749	0.2065	0.2191	0.1947	0.2570	0.2272
LED(16384)	0.4269	0.2862	0.2718	0.3023	0.2831	0.2350	**0.2309**	0.2030	0.2665	0.2236

Figure 3: Results based on Rouge-L Score Metric

Baseline As a baseline, we trained each task by using T5 (small, base, large), BART (base, large, large-cnn), and PEGASUS (xsum, large) models. The performance of each model varied depending on the task. No single model was able to excel at all tasks, and for each task, the model that performed best was different. For “Summary" , PEGASUS-large performed the best showing Rouge-L of 0.4753. For “Holding" , BART-base performed the best showing Rouge-L of 0.2972. For “Facts" , BART-large-cnn performed the best showing Rouge-L of 0.3054. For “Question" , PEGASUS-xsum performed the best showing Rouge-L of 0.3061. For “Conclusion", BART-large performed the best showing Rouge-L of 0.2895. Overall, our experiment shows that the general performance of our baseline model in all tasks is approximately Rogue-L of 0.3.

Larger models As baseline models have maximum token length of 1,024, we have extended our experiment to larger models that can have more maximum token length than the baseline models. LED is capable of at most 16,384 input tokens, and we have experiment with different size of inputs from 1,024 tokens to 16,384 tokens. Unfortunately, LED models showed similar or lower Rouge-L scores compared to the baseline results.

Text preprocessing methods For text preprocessing, we have applied three algorithms independently. Among all three different methods, text ranking outperformed. However, even with this method, the Rouge-L scores did not improve compared to the baseline results.

Applying larger models with text preprocessing As a final approach, we combined the use of larger models with text preprocessing methods. We expected that this combination would improve the performance of the model, but it did not. Instead, this approach resulted in some duplicate summarization sentences and the loss of important sentences, leading to lower Rouge-L scores compared to the baseline predicted sentences.

4.2 Discussion

In this project, we have developed an automated case summarization model that could potentially be used in the legal domain of the World Bank.

As for the baseline results, we were able to produce relatively high Rouge-L scores. Especially for the “Summary" task, the Rouge-L score for our model was 0.4753, indicating that baseline approach can be applied promptly and help reducing human labor. This is due to the fact that the document usually has topic sentences in the front, and therefore small models are still able to produce relatively reasonable predictions.

In order to address the inherent input capacity limit of our baseline BART model, we explored additional approaches, such as using a larger model with an extraction method. However, these approaches did not improve the Rouge scores for any of our summarization tasks. This may be due to the nature of the documents, which contain many similar paraphrased sentences in each paragraph. These paraphrased sentences can sometimes lead the model to think each sentence is separately relevant, causing it to produce the same sentence multiple times. This can result in the exclusion of other important sentences that are actually more relevant to the summary, which can affect the overall quality of the summary produced by the model.

Although our results show relatively high Rouge-L scores, especially for the summary generation task, it is important to note that a high Rouge-L score does not necessarily indicate a high quality of summarization. As such, it is essential that the outputs of our models be evaluated by human annotators or, in the context of this project, law practitioners. Additionally, our model could be improved by using alternative extraction approaches.

5 Conclusions

Although our baseline model shows relatively high Rouge-L scores in summarizing each task, other additional approaches to overcome the capacity limit in BART model did not improve the performance of the model. The model with extraction method could take longer inputs but having more inputs did not increase the Rouge-L socres.

However, we cannot guarantee that a high Rouge-L score from our models' summarization output is directly related to a high quality of the summarized content, as the Rouge score can only calculate the sequences of the overlapping words between the prediction and actual summary that are truncated by the model's max capacity. In other words, a bigger model does not necessarily improve the summarization task of long text documents. Moreover, a high Rouge-L score does not necessarily correlate with a higher quality of summarization. Future work on this project could include having human annotators validate the actual performance of the model in terms of summarization. Additionally, using alternative extraction approaches could potentially improve the performance of the model.

6 Lessons learned

One of the main challenges we faced in this project was our lack of experience with working with real-world datasets. In class, we learn how to build models, but we were not experienced on how to preprocess and prepare data for these models. As a result, we struggled with issues such as converting raw datasets into the appropriate format for use with the Dataset class in Huggingface (`https://huggingface.co/`).

To overcome these challenges, we did our best to learn from online references and resources, and we also reached out to our mentors for guidance and support. Through this process, we were able to solve many of the issues we were facing and gain a better understanding of how to work with real-world data.

One of the key takeaways from this capstone project is that we should not be afraid to ask for help when we need it. Even when working on novel projects, there are always people who have knowledge and experience with both the technical and business aspects of the work, and we can benefit from seeking out their guidance. We plan to carry this lesson with us into future projects.

7 Student contributions

All team members contributed equally to the project. Yoobin Cheong identified a dataset of US Supreme Court cases, conducted Exploratory Data Analysis, and preprocessed the data in preparation for our experiments. Yoon Tae Park built the codebase for our training and evaluation pipeline. Yeong Koh performed detailed analysis of the evaluation results and documented any interesting findings using appropriate visualization methods.

Acknowledgments

This research was supervised and mentored by Kannan Venkataramanan at World Bank, and Najoung Kim, a faculty fellow at CDS.

References

[1] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saeid Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and Krys Kochut. Text summarization techniques: A brief survey, 2017. URL `https://arxiv.org/abs/1707.02268`.

[2] Ahsaas Bajaj, Pavitra Dangati, Kalpesh Krishna, Pradhiksha Ashok Kumar, Rheeya Uppaal, Bradford Windsor, Eliot Brenner, Dominic Dotterrer, Rajarshi Das, and Andrew McCallum. Long document summarization in a low resource setting using pretrained language models, 2021. URL `https://arxiv.org/abs/2103.00751`.

[3] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL `https://arxiv.org/abs/2004.05150`.

[4] Sumit Chopra, Michael Auli, and Alexander M. Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1012. URL `https://aclanthology.org/N16-1012`.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL `https://arxiv.org/abs/1810.04805`.

[6] Mohamed Elaraby and Diane Litman. Arglegalsumm: Improving abstractive summarization of legal documents with argument mining, 2022. URL `https://arxiv.org/abs/2209.01650`.

[7] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019. URL `https://arxiv.org/abs/1910.13461`.

[8] 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. URL `https://aclanthology.org/W04-1013`.

[9] Rada Mihalcea and Paul Tarau. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL `https://aclanthology.org/W04-3252`.

[10] Gianluca Moro and Luca Ragazzi. Semantic self-segmentation for abstractive summarization of long documents in low-resource regimes. URL `https://ojs.aaai.org/index.php/AAAI/article/view/21357`.

[11] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[12] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019. URL `https://arxiv.org/abs/1910.10683`.

[13] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks, 2017. URL `https://arxiv.org/abs/1704.04368`.

[14] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2019. URL `https://arxiv.org/abs/1912.08777`.