

A
Major Project Report
on
**INDIAN LEGAL TEXT SUMMARIZATION FOR LONG
DOCUMENTS**

Submitted in partial fulfilment of the requirements for the award of the
degree of Bachelor of Technology

by
Megha Parate
(20EG105425)



Under the guidance of
Ravinder Reddy B
Assistant Professor,
Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ANURAG UNIVERSITY
VENKATAPUR (V), GHATKESAR (M), MEDCHAL (D), T.S - 500088
TELANGANA
(2023-2024)



CERTIFICATE

This is to certify that the project report entitled **“Indian Legal Text Summarization for Long Documents”** being submitted by **Megha Parate** bearing the Hall Ticket number **20EG105425** in partial fulfilment of the requirements for the award of the degree of the **Bachelor of Technology in Computer Science and Engineering** to **Anurag University** is a record of bonafide work carried out by her under my guidance and supervision from December 2023 to May 2024.

The results presented in this report have been verified and found to be satisfactory. The results embodied in this report have not been submitted to any other University for the award of any other degree or diploma.

Signature of Supervisor
Ravinder Reddy B
Assistant Professor, Dept of CSE
Anurag University

Dr. G. Vishnu Murthy
Professor
Dean, Department of CSE
Anurag University

External Examiner

DECLARATION

I hereby declare that the report entitled “**Indian Legal Text Summarization for Long Documents**” submitted to the **Anurag University** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology (B. Tech)** in **Computer Science and Engineering** is a record of an original work done by me under the guidance of **Ravinder Reddy B, Assistant Professor** and this report has not been submitted to any other university for the award of any other degree or diploma.

Date:

Place: Anurag University, Hyderabad

Megha Parate
(20EG105425)

ACKNOWLEDGMENT

I would like to express our sincere thanks and deep sense of gratitude to project supervisor **RAVINDER REDDY B**, Assistant Professor, Department of Computer Science and Engineering, Anurag University for his constant encouragement and inspiring guidance without which this project could not have been completed. His critical reviews and constructive comments improved my grasp of the subject and steered to the successful completion of the work. His patience, guidance and encouragement made this project possible.

I would like to acknowledge my sincere gratitude for the support extended by **Dr. G. VISHNU MURTHY**, Dean, Department of Computer Science and Engineering, Anurag University. I also express my deep sense of gratitude to **Dr. V. V. S. S. S. BALARAM**, Academic coordinator. **Dr. PALLAM RAVI**, Project Coordinator and project review committee members, whose research expertise and commitment to the highest standards continuously motivated me during the crucial stages of the project work.

I would like to express my special thanks to **Dr. V. VIJAYA KUMAR**, Dean School of Engineering, Anurag University, for their encouragement and timely support in my B. Tech program.

Megha Parate
(20EG105425)

ABSTRACT

Legal judgments are generally very long, and relevant information is often scattered throughout the text. Summarizing a legal judgment requires capturing crucial details comprehensively from the lengthy content. Abstractive-summarization models based on pre-trained language often encounter limitations in handling extended input texts. Furthermore, these models struggle to seamlessly integrate technical terms and specific topics prevalent in legal judgments. A new dataset, recently developed and trained on supreme court case documents, was utilized to maintain relevance and capture legal nuances in the summary. OpenAI's fine-tuned GPT-3.5-turbo model, along with a map-reduce framework, was employed to overcome token limitations. The experimental results revealed a remarkable improvement in rouge scores compared to existing models.

Keywords: GPT-3.5-turbo, Map-reduce, OpenAI, rouge, token.

TABLE OF CONTENT

S. No.	CONTENT	Page No.
	List of Figures	I
	List of Screenshots	II
	List of Tables	III
	List of Abbreviations	IV
1.	Introduction	1
	1.1. Extractive Summarization	1
	1.2. Abstractive Summarization	3
	1.3. Motivation	5
	1.4. Problem Definition	6
	1.5. Problem Illustration	6
	1.6. Objectives of the Project	8
2.	Literature Survey	9
3.	Indian Legal Text Summarization for Long Documents	18
	3.1. Fine-tuning the model	18
	3.2. Generating summaries using fine-tuned model	22
	3.3. Illustration- Map Reduce Paradigm	30
4.	Design	33
	4.1. Use case diagram	33
	4.2. Class diagram	34
	4.3. Activity diagram	35
	4.4. Sequence diagram	36
5.	Implementation	38
	5.1. Functionalities	38
	5.2. Attributes	40
	5.3. Experimental Screenshot	42
	5.4. Dataset	43
6.	Experimental Setup	44
	6.1. Obtain OpenAI API key	44
	6.2. Setup Jupyter Notebook	44
	6.3. Setup Streamlit	46

6.4. Libraries used	47
6.5. Parameters	51
7. Discussion of Results	52
8. Summary, Conclusion and Recommendation	56
9. Future Enhancements	57
10. References	58

List of Figures

Figure No.	Figure Name	Page No.
1.1	Accuracy v/s Input Size	7
2.1	Kernel Similarity Matrix	9
2.2	Cosine Similarity	14
3.1	Overview of fine-tuning procedure	18
3.2	JSON lines dataset	19
3.4	Map-reduce paradigm	22
3.5	Pre-processed document	30
3.6	Chunk 1 of the document	30
3.7	Chunk 2 of the document	31
3.8	Chunk 3 of the document	31
3.9	Summarizing chunks	31
3.10	Combining the summaries	32
4.1	Use case diagram	33
4.2	Class diagram	34
4.3	Activity diagram	35
4.4	Sequence diagram	36
7.1	Precision score of models	53
7.2	Recall score of models	54
7.3	Comparison of the models	55
7.4	Summary Length v/s ROUGE-1 score	58

List of Screenshots

Figure No.	Figure Name	Page No.
3.3	Details of fine-tuned model on OpenAI account	20
5.1	Legal Document	42
5.2	Output	42
5.3	Dataset	43
6.1	Coding environment screenshot	50
6.2	User Interface	50

List of Tables

Table No.	Table Name	Page No.
1.1	Token Limitation problem in existing approach	7
2.1	Comparison of Existing Methods	17
7.1	Average precision score of different models	53
7.2	Average recall score of different models	54
7.3	Average rouge score of different models	55
7.4	Comparison of Summary Length to ROUGE-1 Score	57

List of Abbreviations

Abbreviations	Full Form
GPT	Generative Pre-trained Transformer
NLP	Natural Language Processing
BERT	Bidirectional Encoder Representations Transformer
BART	Bidirectional and Auto-regressive transformer.
IDLE	Integrated Development and Learning Environment
TF-IDF	Term Frequency Inverse Document Frequency
SVD	Singular Value Decomposition
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
LLM	Large Language Model
API	Application Program Interface
LLTS	LLM based Legal Text Summarizer (Proposed Method)
JSON	JavaScript Object Notation
UML	Unified Modeling Language
KESG	Keyword extraction and summary generation.

1. Introduction

In the legal system of India, it takes a very long time to finish cases, and there are more than 4 crore cases [1] still waiting to be resolved. This long delay is a big problem, and it shows that we really need new and smart solutions. People involved in the legal process, like lawyers and judges, find it very hard and time-consuming to summarize lots of papers related to these cases. This is a big issue, and we need to find faster and better ways to handle all this information in the legal system. It's especially important because there are so many pending cases that need attention. So, the idea of creating a special system to summarize legal texts comes into play. This system could make it much easier to deal with the large amount of legal paperwork, transforming how we manage and understand these documents.

Summarization is the process of condensing information from a text or document into a concise form while preserving its essential meaning. The objective is to distill key points and crucial details, making the content more accessible and easily understandable. Two primary types of summarization exist [2]: extractive and abstractive.

1.1. Extractive Summarization

Extractive summarization is a text summarization approach that involves selecting and combining existing sentences or phrases directly from the source text to create a summary. The goal is to extract the most relevant and important information from the original document while preserving the wording and structure of the selected sentences. The procedure is as follow:

Step-1: Sentence Selection: The process begins with identifying key sentences in the source document. These sentences are usually considered informative, containing essential details, facts, or key points.

Step-2: Scoring Sentences: Each sentence is assigned a score based on various criteria, such as the frequency of important words, relevance to the document's main theme, or the presence of keywords. Sentence scoring helps prioritize the selection of sentences for the summary.

Step-3: Ranking Sentences: The sentences are then ranked based on their scores. The highest-scoring sentences are selected for inclusion in the summary. This ranking ensures that the most crucial information is represented in the condensed version.

Step-4: Sentence Combination: The selected sentences are combined to form the extractive summary. The order of sentences may be preserved from the original text, or they may be reorganized to enhance coherence and readability.

Step-5: Output Summary: The final extractive summary comprises a subset of sentences directly lifted from the source document. Since the summary is composed of existing sentences, it maintains the language and style of the original text.

Advantages of Extractive Summarization:

- i. Preservation of Source Information: Extractive summarization retains the wording and structure of the source sentences, ensuring accuracy and fidelity to the original content.
- ii. Grammatical Accuracy: Extracted sentences are grammatically correct and coherent since they are directly taken from the source.
- iii. Simplicity: Extractive methods are generally simpler to implement compared to abstractive methods, as they don't involve generating new content.

Challenges of Extractive Summarization:

- i. Redundancy: Extractive summaries may contain repetitive information present in multiple sentences from the source.
- ii. Coherence Issues: The selected sentences may not seamlessly flow together, leading to potential coherence issues in the summary.
- iii. Limited Creativity: Extractive methods do not generate new expressions or perspectives, limiting the creativity and novelty of the summary.

Extractive Summarization in Legal Documents:

Extractive summarization models for legal documents have been developed as they are simpler to put into action and tend to stay close to the original source. However, these models exhibit certain drawbacks. They may generate summaries that are repetitive, lacking in coherence, or they might overlook crucial details. This limitation arises from their inability to fully capture the nuances inherent in legal language and context. Michalcea et al. [3] introduced a summarization model employing the TextRank algorithm. However, the model exhibits limitations in capturing essential information, relying on word repetition within the text rather than considering the semantic meaning of the words.

1.2. Abstractive Summarization

Abstractive summarization is a text summarization approach that involves generating a condensed version of a document by rephrasing and paraphrasing the content, often introducing new words and phrases not present in the original text. Unlike extractive summarization, which selects and combines existing sentences, abstractive methods aim to produce more concise and coherent summaries that capture the essence of the source material. Detailed Process of abstractive summarization:

Step-1: Understanding the Source Text: The process begins with a thorough understanding of the source document. Natural Language Processing (NLP) techniques are employed to analyse and comprehend the meaning, context, and key concepts within the text.

Step-2: Identifying Key Information: Relevant information and essential concepts are identified within the source document. This involves recognizing the core ideas and extracting key details that contribute to the overall meaning of the content.

Step-3: Generating New Phrases: Unlike extractive summarization, abstractive methods have the capability to generate new words, phrases, and sentences. This involves rephrasing and restructuring the content to create a summary that is more concise while retaining the essential meaning.

Step-4: Paraphrasing and Rewriting: Abstractive summarization involves rewriting and paraphrasing sentences from the source text. This process aims to express the same ideas using different words, improving the overall coherence and flow of the summary.

Step-5: Contextual Understanding: Abstractive methods leverage contextual understanding to ensure that the generated phrases fit well together and maintain the coherence of the summary. This requires a deeper comprehension of the semantic relationships between words and concepts.

Step-6: Incorporating Novel Information: Abstractive summarization allows for the introduction of new information or expressions that may not be explicitly present in the source document. This enhances the creativity and informativeness of the summary.

Step-7: Ensuring Grammatical Correctness: The generated content is adjusted to ensure grammatical correctness and coherence. This involves considering the syntactic structure and linguistic conventions to produce a well-formed and readable summary.

Step-8: Output Summary: The final abstractive summary is a condensed version of the source document, containing newly generated phrases that capture the essential information in a more concise and creative manner.

Advantages of Abstractive Summarization:

- i. Creativity: Abstractive methods can generate novel expressions, providing a more creative and diverse summary.
- ii. Condensation of Information: The ability to rewrite and rephrase allows for a more concise representation of information.
- iii. Improved Coherence: Abstractive summarization often results in summaries with better flow and coherence.

Challenges of Abstractive Summarization:

- i. Factual Accuracy: Ensuring that the generated content is factually accurate can be challenging.
- ii. Handling Ambiguity: Dealing with ambiguous language or complex sentence structures requires advanced natural language understanding.

Abstractive Summarization in legal documents:

The creation of abstractive summarization models for legal documents has encountered obstacles in effectively capturing important information and handling lengthy text. This limitation stems from the unstructured format of legal texts and their intricate nature, characterized by nuanced expressions specific to the legal domain. Consequently, existing models may fail to adequately incorporate and represent these nuanced aspects, leading to a potential loss of critical details during the summarization process. Rush et al. further adopted an abstract text summarization method [4]. They used the abstractive-summarization model to combine and arrange words in the lexicon to form generalized new sentences. Although their method was flexible and conformed to the generation method of natural summarization, it had some issues, such as factual bias and textual repetition.

1.3. Motivation

The motivation to develop an Indian Legal Text Summarizer arises from the palpable challenges experienced by participants in the legal process. The sheer volume of legal documentation, coupled with the complexity of language and nuanced expressions inherent in legal texts, poses a substantial hurdle to the efficient extraction of crucial information. As a consequence, the landscape calls for a novel approach that not only accelerates the summarization process but also ensures the preservation of intricate legal details and nuances unique to the Indian legal system.

Summarization, as a process, holds the potential to revolutionize the legal information handling paradigm. It involves the condensation of lengthy legal texts into concise and informative summaries, thereby facilitating a more accessible and time-efficient comprehension of case-related information. This task can be approached through two broad methodologies: extractive summarization, which involves the direct selection and combination of existing sentences from source texts, and abstractive summarization, which generates condensed versions by rephrasing and paraphrasing content. The choice between these approaches hinges on the specific requirements of the legal task at hand, with each method offering its unique advantages and challenges.

The existing landscape of legal text summarization in India has witnessed the development of extractive summarization models, owing to their simplicity and proximity to the source text. However, these models exhibit drawbacks such as

repetitiveness, lack of coherence, and potential oversight of critical details. In contrast, abstractive summarization, while promising, encounters difficulties in effectively capturing important information due to the unstructured format and nuanced nature of legal texts.

1.4. Problem Definition

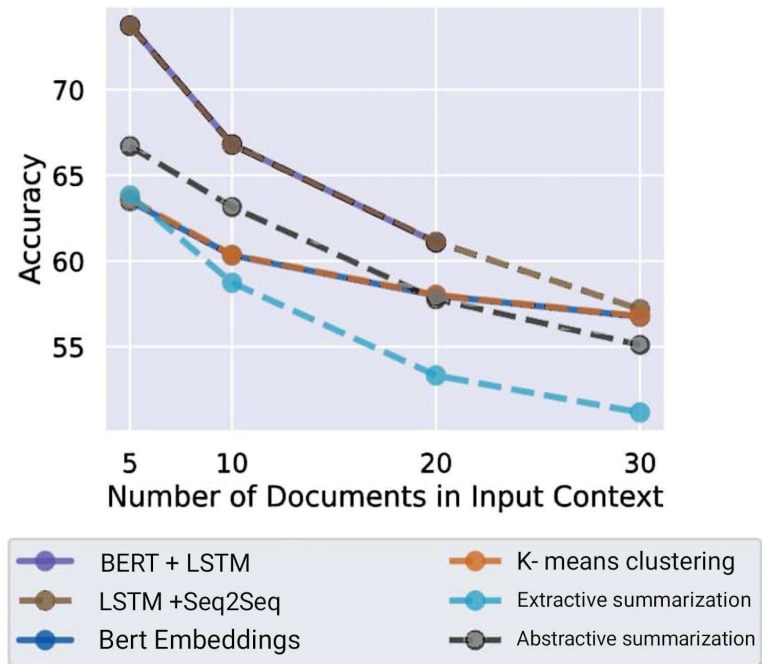
The challenge faced in India's legal system is substantial, primarily stemming from an overwhelming number of court cases that linger unresolved for extended periods. Lawyers and judges, crucial players in the legal process, grapple with the formidable task of managing these cases. Their workload is intensified by the need to extensively read and summarize numerous documents linked to each case, a task that demands both time and effort. The current methods employed for summarization exhibit notable shortcomings. Firstly, there exists a limitation on the number of words or details that can be included in a summary. This constraint poses a challenge to creating concise and clear summaries that capture all essential information. Secondly, the current methods may struggle to fully grasp the intricate legal aspects embedded in the documents, potentially leading to inaccuracies in the generated summaries. These issues underscore the pressing need for an improved system that not only expedites the summarization process but also ensures the accuracy and completeness of the summaries, ultimately alleviating the burdens on legal professionals and enhancing the efficiency of the legal system in India.

1.5. Problem Illustration

In the current approach, summarization relies on the utilization of BERT (Bidirectional Encoder Representations from Transformers). However, the model is constrained by token limitations, meaning it can only generate a restricted amount of text. As the length of the document grows, the comprehensive content of the document increases. Unfortunately, due to the model's token constraints, it cannot generate summaries that exceed this limit. Consequently, it defaults to providing an overall gist of the document, becoming less relevant.

Table 1.1. Token Limitation problem in Existing Approach

S. No.	Token Input	Model token Limitation	Tokens Generated	ROUGE Score
1	754	512	384	0.4351
2	1265	512	415	0.3837
3	2549	512	456	0.2255
4	8457	512	424	0.0515
5	16219	512	453	0.0117



1.1. Accuracy v/s Input size

1.5. Objectives of The Project

The challenges posed by token limitations, potentially restricting the length of our summaries and impeding the comprehensive inclusion of information, coupled with the intricacies in comprehending and capturing legal nuances, prompted the development of an Indian Legal Text Summarizer. This innovative tool has undergone training using a distinctive dataset [5] comprising 7000 judgments from the Supreme Court, accompanied by corresponding abstractive summaries meticulously crafted by legal professionals. The reliability of OpenAI's model stems from its capacity to learn from human feedback, and it distinguishes itself by achieving effective training with a minimal requirement of data – just 50 data pairs [6] proved for the objectives. Recognizing the limitations of conventional approaches, this novel training methodology ensures a nuanced understanding of legal intricacies and addresses constraints on the length of summaries. By leveraging this specialized dataset, our objective is to refine the summarization process, transcending the barriers posed by token limitations and intricacies in legal language, ultimately fostering a more effective and insightful approach to summarizing legal texts in the Indian context.

2. Literature Survey

Summarizing legal texts is hard and takes a lot of time because the documents use complicated language and have tricky details. The words and rules are not easy to understand, making it tough to pick out the important information. S. Ghosh et. al. [1] tackled the disarray in Indian judiciary records by proposing a new method for legal document summarization. They collect, clean, and normalize texts, breaking them into smaller fragments for BART (extractive) and PEGASUS (abstractive) models to summarize. M. C. Popescu et al. [7] approach involves a systematic process where sentences undergo transformation into feature vectors using TF-IDF. This method captures the significance of words based on their frequency and rarity. The resulting vectors are utilized to generate a Kernel Similarity Matrix as shown in Figure. 2.1, assessing the relationships between sentences. The authors [7] then employ Submodular Sentence Ranking, treating summarization as the task of selecting sentences that maximize the summary while avoiding redundancy. This selection process is facilitated through Convex Optimization [8], a technique that strategically picks sentences to create the most effective summary. Ultimately, the chosen sentences are amalgamated to construct the extractive summary, offering a comprehensive overview of the document's key points.

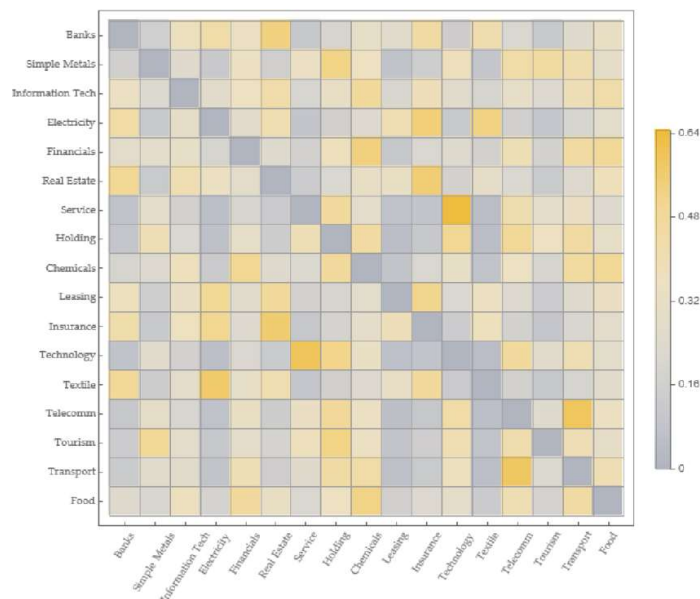


Figure 2.1. Kernel Similarity Matrix

C. Popescu et al. [9] approach begins by converting sentences into feature vectors using Term Frequency Inverse Document Frequency (TF-IDF) [10], which gauges word importance within the document and across the corpus. Then, each sentence gets a salience score based on factors like length, keyword relevance, and potential redundancy. The summarization problem is framed as a convex optimization task, aiming to maximize the sum of salience scores while limiting the summary's length and ensuring sentence weights remain non-negative. To tackle this, the authors employ a coordinate descent-based greedy algorithm, iteratively selecting sentences that best contribute to the overall objective within the specified length constraint. Finally, sentences with non-zero weights post-optimization are included in the extractive summary, forming a concise representation of key content.

A. Nenkova et al. [11] covers various aspects of automatic summarization, starting with different approaches like extractive (selecting key sentences) and abstractive (creating new content) summarization, applicable to both single and multiple documents. It emphasizes effective representation of sentences and documents through features like word frequency, position, and topic coherence. Methods for ranking and selecting sentences, ensuring readability and coherence, and evaluating summary quality using human judgment and metrics like ROUGE scores are discussed. Additionally, it outlines future research directions such as personalization, adapting to different domains, and integration with information retrieval tasks to further enhance automatic summarization techniques.

S. Polsley et al. [12] approach involves several pre-processing steps applied to the document, including the removal of stop words, stemming, and identification of case citations. Keywords are extracted using frequency and a legal term dictionary. Each sentence is then assigned a score, considering various factors such as TF-IDF scores of keywords, legal terms, sentence position, named entities, references to case citations, and readability features. The initial summary is formed by selecting the top-scoring sentences based on a predefined threshold, with additional checks for redundancy and coherence. Furthermore, the system provides an interactive interface that allows users to customize the summary length, highlight sentences related to keywords, and visualize the significance of keywords within the text using heatmaps.

J. W. Yingjie et al. [13] approach involves pre-processing the document and converting it into a matrix where rows represent sentences and columns represent terms based on their frequencies (TF values). By applying Singular Value Decomposition [14] (SVD), this matrix is decomposed into three matrices— U (singular vectors), S (diagonal matrix

of singular values), and V^T (transpose of singular vectors). The most significant singular values and their corresponding vectors are selected from U and V^T , defining the essential latent semantic concepts within the document. Subsequently, each sentence is projected onto this topic subspace, and its relevance to these topics is assessed using cosine similarity, resulting in topic weights. The importance of a sentence is determined by summing the squared topic weights, with higher values indicating greater relevance to the identified topics. Finally, sentences with high importance scores are chosen and combined to form the extractive summary.

B. Samei et al. 's [15] methodology involves transforming multiple documents into a directed weighted graph. In this graph, each sentence serves as a vertex connected by edges that represent semantic similarity. The weights of these edges indicate the level of similarity between sentences. Distortion measures, such as Kullback-Leibler divergence [16], quantify the information loss when replacing one sentence with another similar one. An iterative ranking algorithm is applied to this graph. Initially, sentences are ranked based on their inherent information and distortion due to neighboring sentences. In each iteration, sentences update their ranks by considering their own rank, their neighbors' ranks, edge weights, and distortion measures until the ranks stabilize. The sentences with the highest final ranks, reflecting importance and minimal distortion, are then chosen for inclusion in the ultimate summary.

A. Joshi et al. [17] Summocoder introduces an unsupervised approach for extractive text summarization that utilizes three key metrics in sentence selection. First, it measures sentence relevance based on content using a deep auto-encoder network that reconstructs sentences from compressed representations; lower reconstruction errors indicate higher relevance. Second, it gauges sentence novelty by assessing the similarity between sentence embeddings in a semantic space, favoring sentences dissimilar to previously chosen ones for their novelty and informativeness. Lastly, it incorporates sentence position relevance, a manually designed feature that assigns greater weight to initial sentences while dynamically adjusting weights based on the document's length, recognizing the importance of sentences at the document's outset.

V. Parikh et al. [18] Lawsum introduces a dataset with 10,000 Supreme Court judgments and paired summaries. Pre-processing ensures accuracy by eliminating noise, normalizing legal abbreviations, and categorizing sentences by their "rhetorical role." Using handcrafted summaries, Lawsum labels sentences in the original documents as "summary-worthy" or "non-summary" automatically, streamlining model training without extensive manual work.

A supervised classification model, considering features like sentence length, rhetorical role, position, and keyword frequency, predicts the likelihood of a sentence being "summary-worthy." The final summary is crafted by extracting top-scoring sentences based on the model's predictions, guided by a predefined threshold. Nguyen et al. [19] present an advanced document-level encoder, building on BERT (Bidirectional Encoder Representations from Transformers), to capture deep semantic connections among sentences. This innovative architecture combines multi-layer Bi-LSTMs [20] with BERT representations, merging local sentence details with broader document context. For extractive summarization, inter-sentence Transformer layers identify and select informative sentences. Abstractive summarization involves a dedicated decoder network using attention mechanisms, crafting new sentences with the document-level encoder. To address disparities, a unique fine-tuning schedule with separate optimizers and varied learning rates is introduced for stability and improved summary quality. A two-staged fine-tuning process refines abstractive summarization, optimizing the encoder representation first and then fine-tuning the decoder, resulting in enhanced summarization performance. Siyao Li et al. [21] employ a standard encoder-decoder setup with an LSTM encoder guiding a Transformer decoder to generate summaries. Unlike traditional ROUGE-L rewards, the system integrates Word Mover's Distance (WMD) [22] for semantic similarity, enhancing coherence. Policy Gradient Reinforcement Learning is used, with the agent interacting based on WMD rewards, updating decoder parameters through REINFORCE or A2C algorithms for improved summary generation. A curriculum learning strategy is introduced to counter exposure bias, starting with simpler documents and progressing to complex ones, refining the system's summarization abilities. K. Agrawal [23] applies existing extractive summarization techniques to legal case summaries, without introducing new methods. The paper explores the adaptation of two established techniques: TF-IDF with Sentence Position Scoring, evaluating sentences based on term frequency and position, and LexRank, a graph-based algorithm identifying sentences with high centrality scores for inclusion. These methods are applied after standard preprocessing steps, tailored for legal case documents by removing citations and unnecessary formatting. Yang Liu et al. [24] introduce a novel document-level encoder based on BERT, combining multi-layer Bi-LSTMs with BERT (Bidirectional Encoder Representations from Transformers) representations for comprehensive sentence understanding. For extractive summarization, stacked inter-sentence Transformer layers discern

informative sentences. Abstractive summarization involves a separate decoder network using attention mechanisms with the document-level encoder. To address mismatches, a unique fine-tuning schedule employs distinct optimizers and learning rates for stability. A two-stage fine-tuning process optimizes the encoder representation first and then refines the decoder for enhanced abstractive summarization. M. Norkute et al. [25] utilized a pre-trained BART model for legal text summarization, fine-tuned with a dedicated legal dataset. To enhance model interpretability, they incorporated attention scores to highlight influential sections in the source document and source attribution to identify key sentences. Evaluation involved participants assessing summaries with and without these features, including tasks like answering legal questions and evaluating summary quality. Feedback through questionnaires gauged trust in the system, understanding of summaries, and perceived workload. K. Merchant et al. [26] likely utilizes Natural Language Processing (NLP) techniques, including tokenization, stemming, lemmatization, and part-of-speech tagging, to prepare legal text for analysis. The paper may also leverage Latent Semantic Analysis (LSA) [27] as a statistical technique to identify hidden semantic relationships between words and concepts. Additionally, various summarization models, such as extractive summarization (selecting key sentences) or abstractive summarization (generating new sentences capturing main points), could be employed. A. Farzindar et al. [28] LetSum uses a structured approach in its thematic analysis of legal documents, identifying Introduction, Context, Juridical Analysis, and Conclusion themes. Employing a rule-based system and a legal concept dictionary, the process prioritizes sentences with theme-relevant keywords and those in proximity to theme markers. Redundancy is addressed by identifying and removing duplicate or irrelevant sentences. The final summary is generated by rearranging selected sentences to maintain coherence and a smooth flow. S. S. Megala et al. [29] collected legal documents from an unspecified website during their study, lacking detailed information on document types or lengths. The pre-processing phase involved standardization, converting documents to lowercase, removing punctuation, and eliminating stop words. Feature extraction employed two approaches: keyword extraction, identifying frequent terms in abstracts and sentences with high cosine similarity scores to abstracts; and a network approach, representing the document as a sentence network with sentence importance determined by degree centrality and Jensen-Shannon divergence [30]. Summary creation involved selecting sentences based on importance scores. Evaluation used the Cosine Measure

as shown in figure. 2.2 to compare similarity and the Jensen-Shannon Distance to quantify information difference between the summary and the original document.

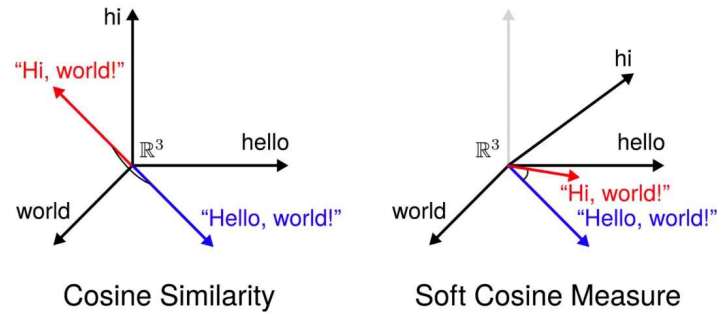


Figure 2.2. Cosine Similarity

Addressing factual biases and textual repetition concerns, See et al. [31] introduced a pointer-based seq2seq model, mitigating the challenges associated with unrecorded and rare words to some extent. They also implemented a coverage mechanism to handle sequence duplication. Despite the improved readability of the generated summaries with their seq2seq model, a substantial corpus was necessary for training. Limited availability of large volumes of legal judgment documentation and reference summaries due to legal restrictions posed a constraint. Feijo et al. [32] introduced the LegalSumm method, leveraging a pre-training model. LegalSumm addressed lengthy legal judgments by creating various perspectives on the source text and incorporated a concealed module to assess the authenticity of candidate summaries in relation to the source text. However, their approach relied on the RulingBR dataset, which is in Portuguese and characterized by shorter documentation and summaries compared to typical legal judgments. Consequently, the challenge of lengthy legal judgments remained partially unresolved. Ilias Chalkidis et al. [33] present a new English legal judgment dataset from the European Court of Human Rights (ECHR), encompassing case facts, relevant articles from international human rights treaties, and final decisions. The study evaluates various neural models, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and attention-based models like BERT, trained on the ECHR dataset. Tasks include binary violation classification, multi-label classification for identifying violated articles, and case importance prediction for landmark cases. Model performance is assessed using standard metrics such as accuracy, precision, recall, F1-score for classification tasks, and mean squared error for regression tasks. A. Sleimi et al. [34] conduct a thorough analysis of semantic legal

metadata types in requirements engineering, aiming to align diverse definitions for legal document analysis. Their approach utilizes Natural Language Processing (NLP) techniques, including tokenization, part-of-speech tagging, and named entity recognition, to identify terms representing specific semantic legal metadata types. These terms are categorized using predefined rules within the NLP framework, while more complex metadata types are addressed with supervised and semi-supervised machine learning techniques, such as Support Vector Machines (SVMs), to improve extraction accuracy. The approach's effectiveness is assessed through two case studies involving real-world legal documents from Luxembourgish legislation, comparing the extracted metadata with manually annotated data to evaluate precision and recall. C. Prasad et al. [35] utilize a curated dataset of cleaned news articles and Wikipedia abstracts, with segmented sentences paired with summaries. Employing a sequence-to-sequence (Seq2Seq) model, the encoder processes input text to create a hidden representation, and the decoder generates the summary sentence by sentence. An attention mechanism within the decoder enhances focus on relevant parts of the input text during summary generation. The model is trained using the Adam optimizer and cross-entropy loss function, and evaluation is based on ROUGE scores, assessing the overlap between generated and reference summaries to gauge model performance. Hachey Ben et al. [36] developed a system for analyzing judgments from the UK House of Lords. The corpus includes annotated rhetorical functions for each sentence, with features at the sentence level, such as length, position, cue phrases, and rhetorical functions like "summary" or "argument." Document-level features include document length and citations per sentence. Two classifiers, a rhetorical and a summary classifier, are trained using Support Vector Machines (SVMs), incorporating features like term frequency-inverse document frequency (TF-IDF) and sentence position. The summarization process selects top-ranked sentences from the summary classifier, prioritizing relevance and diversity, with high rhetorical importance guiding the final summary. Farzindar Atefeh et al. [37] initiate document summarization by segmenting the legal document thematically using keywords and discourse markers. Each sentence within these segments is assigned an argumentative role (introduction, context, legal analysis, or conclusion) for core point identification. Importance scoring follows, considering factors like keyword frequency, centrality within the thematic segment, and role in the argumentative structure. The highest-scoring sentences are then extracted and consolidated into a table-style summary, prioritizing coherence and readability.

while aiming to preserve the original wording when possible. This approach distinguishes itself from traditional extractive summarization methods. V.R. Kumar et al. [38] propose an innovative legal document summarization method utilizing Latent Dirichlet Allocation (LDA) [39], a topic modeling technique. The approach involves preprocessing, removing irrelevant information like headers, footers, and legal jargon, followed by sentence extraction and segmentation. LDA is applied to reveal latent topics within the document, assigning probabilities to sentences for belonging to these topics. The subsequent step selects sentences with high probabilities for the most relevant topics, considering ranking criteria like topic coherence and sentence centrality. The final stage involves generating a concise summary by ordering and concatenating the selected sentences. M. Saravanan et al. [40] collect data from 50 Supreme Court judgments in India across various domains. Legal experts manually assign rhetorical roles, such as background information, facts, reasoning, legal precedent, and conclusion, to each sentence. Feature extraction involves 75 linguistic features, covering aspects like sentence length, grammatical features, part-of-speech tags, and named entity recognition. Two machine learning models are trained: a Conditional Random Field (CRF) [41] for automatic identification of rhetorical roles based on features and the previous sentence's role, and an Extractive Summarization System that selects sentences with high importance based on rhetorical roles, prioritizing conclusions and legal precedents. Evaluation includes assessing accuracy in identifying roles and the quality of generated summaries using ROUGE metrics for automatic summarization evaluation.

Table 2.1. Comparison of Existing Methods

Sl.no	Author (s)	Method	Advantages	Disadvantages
1	N. Begum and A. Goyal [36]	Analysis of Legal Case Document Automated Summarizer	Efficiently summarizes paragraphs and provides Accessibility	Lack of accuracy and Cannot summarize lengthy document
2	Parikh et. al [13]	LawSum: A weakly supervised approach for Indian Legal Document Summarization	It is Aware of the Structure of summary generated	Focuses only on those sentences which contain more important words
3	Moro G. et. al [37]	Multi-language transfer learning for low-resource legal case summarization	Extracts critical information Efficiently	Cannot summarize lengthy documents
4	Kore et. al [38]	Legal Document Summarization Using NLP and ML Techniques	It provides both abstractive and extractive summaries	Does not address specific requirements of different legal sub-domains
5	S. Ghosh et. al [1]	Indian Legal Text Summarization: A Text Normalization-based Approach	Uses pretrained models like BART, PEGASUS	Cannot summarize lengthy documents

3. Indian Legal Text Summarization for Long Documents

The proposed methodology aims to improve the ROUGE score compared to existing methods [33][34][22][17][35], indicating that the generated summary effectively captures the essence of the entire document and also aims to overcome token limitation problem by employing MapReduce method. The process comprises two distinct phases: the initial phase involves fine-tuning the model, and the subsequent phase entails generating summaries utilizing the fine-tuned model.

3.1. Fine-tuning the model

Fine-tuning enhances the capabilities of models accessible through the API by delivering superior results compared to prompting, accommodating training on a larger volume of examples beyond prompt constraints, achieving token savings through concise prompts, and enabling requests with lower latency. Below flowchart in Figure 3.1. gives a high-level overview of fine-tuning process.

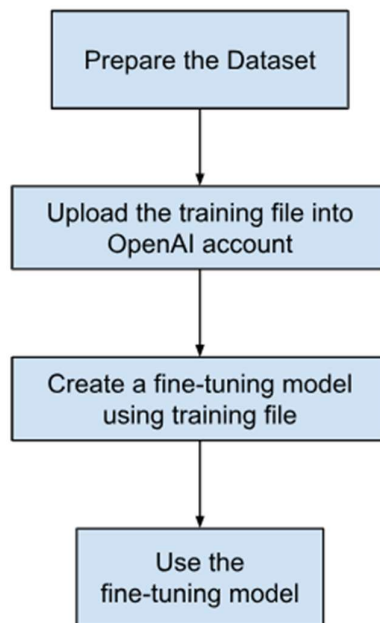


Figure 3.1. Overview of fine-tuning procedure.

The fine-tuning process encompasses the following stages:

3.1.1 Prepare and upload training data:

Download the dataset [4] which contains more than 7000 supreme court judgements and its corresponding summaries. The dataset needs to be structured like a conversation in the same way as the Chat Completions API. This means it should be a list of messages, where each message includes a role (like who is speaking) and the actual content of the message. To make this happen, the dataset must be transformed into a JSON lines file format.

For example:

```
{"messages": [{"role": "system", "content": "You are a legal text summarizer. You should help user to summarize his legal documents (judgements)."}, {"role": "user", "content": judgement}, {"role": "assistant", "content": summary}]}
```

The system will instruct on what to do with the data provided by the user to the assistant.

We created the dataset in JSON lines format as shown in Figure 3.2.

```
{"messages": [{"role": "system", "content": "You are a legal text summarizer. You should help user to summarize his legal documents (judgements)."}, {"role": "user", "content": "ION: Criminal Appeal 89 of 1961.\nAppeal by special leave from the judgment and order dated December 8, 1960, of the Allahabad High Court in Criminal Appeal No. 1782 of 60 and Referred No. 125 of 1960, section K. Kapur, for the appellant, G. C. Mathur and C. P. Lal, for the respondent.\nDecember 19.\nThe Judgment of the Court was delivered by RAGUHBAR DAYAL, J.\nRam Singh appeals, by special leave, against the order of the Allahabad High Court dismissing his appeal and confirming 204 his conviction and sentence of death, under section 302, I.P.C., by the Session Judge, Etawah.\nThe prosecution case, in brief, is that due to enmity, the appellant caused injuries to Sheo Sahai, who was sleeping in his cattle shed in village Bhadurpur Ghar, with a sword at about mid night on the night between June 14 15, 1960.\nSheo Sahai died of the injuries received.\nThe appellant thereafter proceeded to the Canal Distributory at some distance from the village and
```

Figure 3.2. JSON lines dataset

After preparing the training data, the next step is to upload it using the Files API. This allows the data to be utilized in fine-tuning jobs effectively. Use the following steps:

Step-1: Import OpenAI module.

Step-2: Create an instance of the OpenAI client for interactions.

Step-3: Open the JSON Lines dataset in binary read mode.

Step-4: Use the OpenAI client to create a new file by uploading the opened file.

Step-5: Set the purpose of the file as "fine-tune."

3.1.2. Create a Fine Tune Model:

After uploading the file, the subsequent action is to initiate a fine-tuning job. This is accomplished by creating a fine-tuning job through the OpenAI SDK:

Step-1: Create an instance of the OpenAI client for interactions.

Step-2: Utilize the OpenAI client to create a fine-tuning job.

Step-3: Specify the training file ID that was returned when the training file was uploaded to the OpenAI API as its identifier (e.g., "file-abc123")

Step-4: Indicate the model to be used for fine-tuning as "gpt-3.5-turbo."

Upon commencing a fine-tuning job, the time for completion may fluctuate based on queue dynamics and the duration of the training process, which is contingent on factors like the model and dataset size. After the training concludes, users will be notified with a confirmation mail.

Once the job has been successfully completed, you'll observe that the fine-tuned model field is filled with the model's name when retrieving the job details. And even the fine-tuned model will be reflected in your OpenAI as shown below in Figure 3.3.



Figure 3.3. Details of fine-tuned model on OpenAI account

The overall algorithm for fine tuning:

Step 1: Import pandas

Step 2: Upload the csv dataset

Step 3: Encode the dataset with 'cp1252'

Step 4: Create a new jsonl file named 'training.jsonl' and open it in writing mode

Step 5: Create a function `CREATE_DATASET` which takes a judgment and its corresponding summary as the input parameters.

Step 6: The function creates a message as shown in the above example and returns it.

Step 7: Iterate through the CSV file and send the judgment and summary one by one to `CREATE_DATASET`.

Step 8: While iterating write the returned message (every time in a new line) from `CREATE_DATASET` in training.jsonl.

Step 9: From 'openai' library import 'OpenAI' class

Step 10: Create an instance of the OpenAI class as 'client' with an 'api_key' parameter set as user API KEY, which will be used to communicate with the OpenAI API.

Step 11: Upload the training file using 'file.create()' method under 'OpenAI' and set its 'file' parameter as open 'training.jsonl' file in read binary mode and set the 'purpose' parameter as 'fine-tune'. A file id will be generated

Step 12: Fine tune the model using 'fine_tuning.jobs.create()' method under OpenAI class and set its parameters. Set 'training_file' as the file id generated after uploading the training file, set the 'model' as the 'gpt-3.5-turbo'. This will create a fine-tuning job id.

Step 13: Retrieve the fine-tuned model status using `fine_tuning.job.retrieve()` method under OpenAI. The model name will be generated.

3.2. Generating summaries using fine-tuned model:

In our legal document summarization process, we follow the Map-Reduce paradigm as illustrated in Figure 3.4., breaking it down into two key steps. In the Map step, each legal document is treated individually using the fine-tuned model, it dissects complex legal language, providing concise summaries for each document section. These

individual summaries serve as the building blocks for our summarization process. In the Reduce step, we assemble these summaries into a cohesive whole, creating a comprehensive summary that captures the essence of the entire legal document. To address potential length challenges in the individual summaries, we introduce a compression step, ensuring a smooth and coherent summarization process.

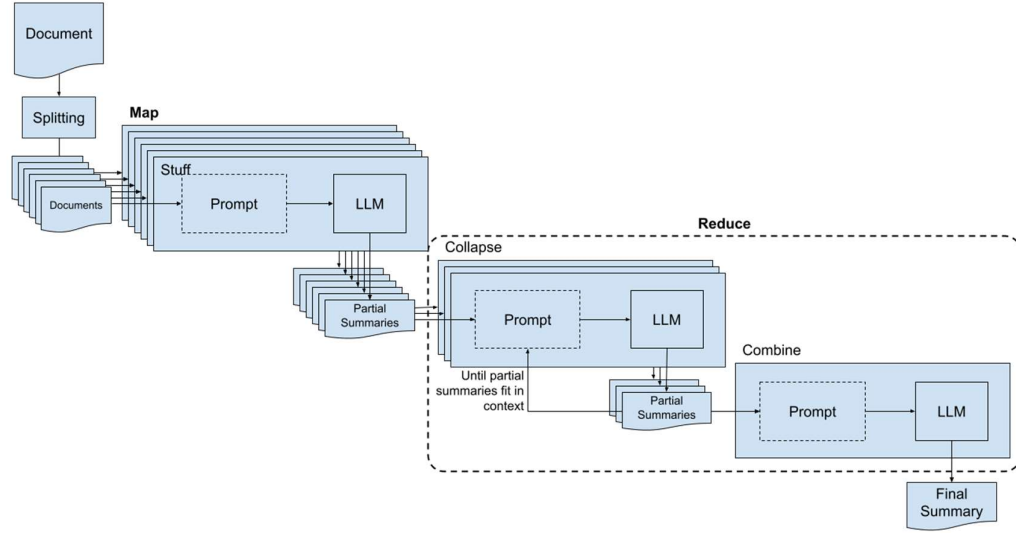


Figure 3.4. Map-reduce paradigm

3.2.1 Install python libraries:

During the initial phase of workflow, we proceed with the installation of three Python libraries:

- i) *OpenAI*: Incorporated to utilize the robust language models provided by OpenAI for the purpose of summarizing legal documents.
- ii) *LangChain*: Crucial for the efficient implementation of document mapping, reduction, and combination workflows.
- iii.) *Tiktoken*: Helps manage token counts within text data, ensuring efficient usage of language models and avoiding token limit issues.

3.2.2. Initialize the OpenAI LLM:

To leverage advanced language models for summarization, follow the outlined procedure:

Step-1: Import the necessary libraries, including OpenAI for interfacing with the OpenAI API and ChatOpenAI from the langchain.chat_models module.

Step-2: Define the API key for authentication with the OpenAI API.

Step-3: Create an instance of the ChatOpenAI class with specified parameters, such as temperature, maximum tokens, API key, and the model to be used.

3.2.3. Splitting the text into chunks:

The Text Splitter addresses the token limit challenge by dividing the text into smaller segments, each staying within the token limit. This guarantees efficient processing by the language model, preventing token capacity breaches.

Step-1: Import the CharacterTextSplitter module from langchain.text_splitter.

Step-2: Initialize the Text Splitter using Tiktoken encoder and specify the chunk_size and chunk_overlap.

The inclusion of the "chunk_overlap" parameter ensures there's some overlap between the chunks, safeguarding against any potential loss of information during the splitting process.

3.2.4. Loading the legal document:

The purpose of this loader is to handle the loading and processing of the legal file's contents. Subsequently, it loads and partitions the legal document into document chunks. These chunks are likely indicative of distinct sections or pages within the legal file. Ultimately, the function yields a list containing these chunks, rendering them accessible for subsequent summarization tasks:

Step-1: Import the PyPDFLoader class from the langchain.document_loaders module.

Step-2: Define a function named chunks that takes a parameter pdf_file_path, representing the file path of the legal document to be processed.

Step-3: Inside the function:

- a. Create an instance of the PyPDFLoader class named "loader" by passing the provided pdf_file_path.
- b. Use the "loader" to load and split the legal document into smaller document chunks.
- c. Store the resulting document chunks in a variable named "docs."
- d. Return the list of document chunks ("docs") as the output of the function.

3.2.5. Import libraries for map-reduce paradigm:

- i) Import the MapReduceChain class from the langchain.chains.mapreduce module, which is essential for implementing the MapReduce paradigm in LangChain.
- ii) Import the CharacterTextSplitter class from the langchain.text_splitter module, allowing for text splitting, which is often a necessary preprocessing step in document analysis.
- iii) Import the ReduceDocumentsChain and MapReduceDocumentsChain classes from the langchain.chains module. These classes are integral for reducing and mapping documents in the LangChain MapReduce workflow.
- iv) Import the PromptTemplate class from the langchain module, providing a template for generating prompts, which is fundamental for interacting with language models.
- v) Import the LLMChain class from the langchain.chains module, which is crucial for orchestrating language model interactions within the LangChain framework.
- vi.) Import the StuffDocumentsChain class from the langchain.chains.combine_documents.stuff module, facilitating the combination of documents in the LangChain workflow.

3.2.6. Define Templates and creating LLM Chains:

The code establishes two templates, namely map_template and reduce_template. These templates act as structured instructions, guiding a language model on the procedures to follow when processing and summarizing collections of documents. The instructions are:

```
map_template = """"The following is a set of documents
```

```
{docs}
```

Based on this list of legal docs, Extract the most important sentences and summarize and highlight key entities related to the legal document, such as section number, names of plaintiffs, locations, and other pertinent details. The summary length should vary to encompass the essence of the document. Helpful Answer:"""

```
reduce_template = """The following is set of summaries:
```

```
{doc_summaries}
```

Take these and distill it into a final consolidated summary and retain all the points of the summaries, the summary length can vary to encompass the essence of the provided summaries. Helpful Answer:"""

Creating an instance of the PromptTemplate class involves using the template provided in the variable 'reduce_template'. This instance is then used to initialize a processing chain (LLMChain) that incorporates both a language model (fine-tuned model) and the generated prompt (reduce_prompt).

Two LLMChains, namely map_chain and reduce_chain, are equipped with these templates to perform the mapping and reduction stages within the document summarization process. This configuration enhances the structure and manageability of the summarization workflow.

3.2.7. LLM Chains for Map-reduce paradigm:

After preparing a map_chain and reduce_chain These facilitate in-depth summarization and reduction of each document within the sequence. The procedure is as follows:

Step-1: Creating an instance of the StuffDocumentsChain by specifying 'reduce_chain' as a language model chain and a variable which stores the chain of summaries.

Step-2: Create an instance of the ReduceDocumentsChain by providing the chain of documents obtained from Step-1, and collapse or compress the chain to overcome the limit of token input.

Step-3: Create an instance of the MapReduceDocumentsChain by specifying map_chain as mapping operation, reduce document chain obtained in previous step as reduce operation, and assign the generated summary to a variable.

The Combine Documents Chain (`combine_documents_chain`) is vital in document summarization, merging individual legal document summaries from the "Map" step into a cohesive text string, named "doc_summaries," for subsequent processing in the "Reduce" step.

The Reduce Documents Chain (`reduce_documents_chain`) concludes the summarization process by taking the combined document string and performing thorough reduction and summarization. To address token limit issues, the chain recursively compresses lengthy documents into smaller chunks, limiting each to 5,000 tokens for efficient processing.

Following the MapReduce paradigm, the Map-Reduce Documents Chain (`map_reduce_chain`) employs the `map_chain` in the "Map" step to process each legal document, producing initial summaries. In the "Reduce" step, the chain utilizes `reduce_documents_chain` to merge these initial summaries into a final, comprehensive document summary, stored in the "docs" variable within the LLM chain.

3.2.8. Create a summarization function:

Defines a function `summarize_pdf` that takes a `file_path` as an input parameter. The function utilizes a `text_splitter` to split the contents of the PDF document located at the specified `file_path` into distinct chunks. These chunks are then processed using a `map_reduce_chain`, which seems to be a document summarization mechanism based on a map-reduce paradigm. The final result, representing the summarized content of the PDF, is obtained by running the `map_reduce_chain` on the split documents. The result is stored in the variable `result_summary`. Finally, the code prints the summarized content.

The overall algorithm of summarization:

Step 1: Install OpenAI, LangChain, Tiktoken, and PyPDF.

Step 2: Import the OpenAI library and assign the user's API key to the 'API_KEY' variable.

Step 3: From the 'OpenAI' library, import the 'ChatOpenAI' class.

Step 4: Initialize an instance of the ChatOpenAI class as 'llm' with parameters. Set 'temperature' to 0 to keep the output more focused and less random, set 'max_tokens' to 1000 to process the input within that limit, set 'openai_api_key' to API_KEY to access the OpenAI API, and set 'model' to the model name obtained after fine-tuning.

Step 5: Import the 'CharacterTextSplitter' class from the 'text_splitter' module in the LangChain package.

Step 6: Initialize this class with a 'text_splitter' object and with specific parameters. Set 'chunk_size' to 1000 tokens, 'chunk_overlap' to 50 to avoid missing information between chunks, and 'separator' to '\n\n' (double newline) to divide the chunks into paragraphs.

Step 7: Import the 'PyPDFLoader' class from the 'document_loaders' module of the LangChain library.

Step 8: Define a function 'file_loader' with the input parameter as the legal document file path.

Step 9: Initialize the 'PyPDFLoader' class with a 'loader' object using the legal document file path as a parameter in the function. This will access the PDF.

Step 10: Call the 'load' method using the 'loader' object to extract text from the PDF and assign the result to the 'docs' variable.

Step 11: The 'file_loader' function returns 'docs'.

Step 12: Import the 'MapReduceChain' class from the 'mapreduce' module in the LangChain package.

Step 13: Import the 'CharacterTextSplitter' class from the 'text_splitter' module in the LangChain package.

Step 14: Import the 'ReduceDocumentsChain' and 'MapReduceDocumentsChain' classes from the 'chains' module in the LangChain package.

Step 15: Import the 'PromptTemplate' class from the LangChain package.

Step 16: Import the 'LLMChain' class from the 'chains' module in the LangChain package.

Step 17: Import the 'Stuff' class from the 'chains.combine_documents' module in the LangChain package.

Step 18: Define the 'map_template' by providing instructions on how to process the received chunks of documents.

Step 19: Create a 'map_prompt' by utilizing the 'from_template' method from the 'PromptTemplate' module. Pass the 'map_template' as a parameter to transform this template into a prompt.

Step 20: Construct an LLM Chain to guide the LLM model by sending the prompt to the LLM. Utilize the 'LLMChain' class, specifying the parameters 'llm' as the fine-tuned model name and 'prompt' as the 'map_prompt'.

Step 21: Define the 'reduce_template' by outlining the instructions for processing the received summaries of the document chunks and converting them into a single concise summary.

Step 22: Create a 'reduce_prompt' by utilizing the 'from_template' method from the 'PromptTemplate' module. Pass the 'reduce_template' as a parameter to transform this template into a prompt.

Step 23: Construct an LLM Chain to guide the LLM model by sending the prompt to the LLM. Utilize the 'LLMChain' class, specifying the parameters 'llm' as the fine-tuned model name and 'prompt' as the 'reduce_prompt'.

Step 24: Construct the chain (combine_documents_chain) for combining the document summaries using the 'StuffDocumentChain' class. Specify the parameters 'llm_chain' as 'reduce_chain' and 'document_variable_name' as 'doc_summaries,' which will be passed to the 'llm_chain'.

Step 25: Construct the chain for generating a concise summary of the document using ReduceDocumentsChain class. Specify the parameters 'combine_documents_chain' as 'combine_documents_chain' which will combine the document into a string and summarize it, 'collapse_documents_chain' as 'combine_documents_chain' which will maintain the length of the summary and summarize it recursively, 'max_token' as 20% of number of tokens in legal document.

Step 26: Construct a chain for implementing a MapReduce workflow for processing documents using the 'MapReduceDocumentsChain' class. Specify the parameters 'llm_chain' as 'map_chain' which will summarize the individual chunks of the document, 'reduce_documents_chain' as 'reduce_documents_chain' which will combine the outputs of the 'map' phase, 'document_variable_name' as 'docs' which acts as the input variable for map phase, 'return_intermediate_steps' as 'False' which specifies that only the final output should be returned, not intermediate results from the "map" or "reduce" steps.

Step 27: Define a function named `summarize_pdf` that takes a file path as input and returns a summary of the PDF document.

Step 28: Call the 'file_loader' function and provide the file path as its argument, which will then return the 'docs.' Subsequently, set this 'docs' as an input parameter for the 'split_documents' method of the 'text_splitter' class (defined in step 6). The 'split_documents' method returns the chunks of the document, assigned to 'split_docs.'

Step 29: Execute the 'map_reduce_chain' using the 'run' method and provide 'split_docs' as an input parameter. This will perform the complete map-reduce process and return the final summary.

Step 30: Define 'file_path' as the input document's file path.

Step 31: Call the 'summarize_pdf' function, using the 'file_path' as an input, and store the result in the 'result_summary' variable.

Step 32: Print the 'result_summary.'

3.3. Illustration-Map Reduce Paradigm

Map reduce paradigm was used to overcome the problem of token limitation. The following stages provide you with an example of how the document will be summarized throughout the process.

1. ***Load and process the Legal Document:*** Extract the content from the legal document.

This case (Appeal No. LXVI of 1949) involves an appeal against a judgment from the High Court of Judicature in Bombay regarding the deductibility of municipal property tax and urban immoveable property tax under section 9(1)(iv) of the Indian Income Tax Act. The appellant, an investment company in Bombay, claimed deductions for both taxes, arguing that they are annual charges assessed on the property's annual value. The court analyzed the relevant statutes, including the City of Bombay Municipal Act, 1888, and the Bombay Finance Act, 1932, determining that the liability for these taxes is indeed annual and recurrent, making them eligible for deduction under section 9(1)(iv). The court rejected the argument that the taxes were contingent or variable, emphasizing their annual nature.

In its decision, the Supreme Court reinstated the appellant's claim, holding that municipal property tax and urban immoveable property tax are allowable deductions under section 9(1)(iv). The court highlighted the annual liability of the taxes, as determined at the beginning of each official year, and rejected the notion that they are contingent charges. The decision clarified that if municipal taxes fall within the scope of clause (iv), deduction is claimable. The court's analysis emphasized the recurring nature of the taxes and their alignment with the language of the statute, ultimately allowing the appeal and awarding costs to the appellant.

This case underscores the importance of interpreting tax laws in alignment with the specific language used in statutes. The court's decision focused on the annual and recurrent nature of the property taxes in question, emphasizing their eligibility for deduction under the relevant provision of the Indian Income Tax Act.

Figure 3.5. Preprocessed document

Token count: 5078

2. ***Divide the Documents into chunks:*** Divide the documents into paragraph with token count less than 4000 to eliminate the token limitation problem.

Chunk 1:

This case (Appeal No. LXVI of 1949) involves an appeal against a judgment from the High Court of Judicature in Bombay regarding the deductibility of municipal property tax and urban immoveable property tax under section 9(1)(iv) of the Indian Income Tax Act. The appellant, an investment company in Bombay, claimed deductions for both taxes, arguing that they are annual charges assessed on the property's annual value. The court analyzed the relevant statutes, including the City of Bombay Municipal Act, 1888, and the Bombay Finance Act, 1932, determining that the liability for these taxes is indeed annual and recurrent, making them eligible for deduction under section 9(1)(iv). The court rejected the argument that the taxes were contingent or variable, emphasizing their annual nature.

Figure 3.6. Chunk 1 of the document

Token count: 2641

Chunk 2:

In its decision, the Supreme Court reinstated the appellant's claim, holding that municipal property tax and urban immoveable property tax are allowable deductions under section 9(1)(iv). The court highlighted the annual liability of the taxes, as determined at the beginning of each official year, and rejected the notion that they are contingent charges. The decision clarified that if municipal taxes fall within the scope of clause (iv), deduction is claimable. The court's analysis emphasized the recurring nature of the taxes and their alignment with the language of the statute, ultimately allowing the appeal and awarding costs to the appellant.

Figure 3.7. Chunk 2 of the document

Token count: 1846

Chunk 3:

This case underscores the importance of interpreting tax laws in alignment with the specific language used in statutes. The court's decision focused on the annual and recurrent nature of the property taxes in question, emphasizing their eligibility for deduction under the relevant provision of the Indian Income Tax Act.

Figure 3.8. Chunk 3 of the document

Token count: 573

3. *Send all the chunks for summarization:* Summarize all the chunks parallelly to obtain a summary of each chunk.

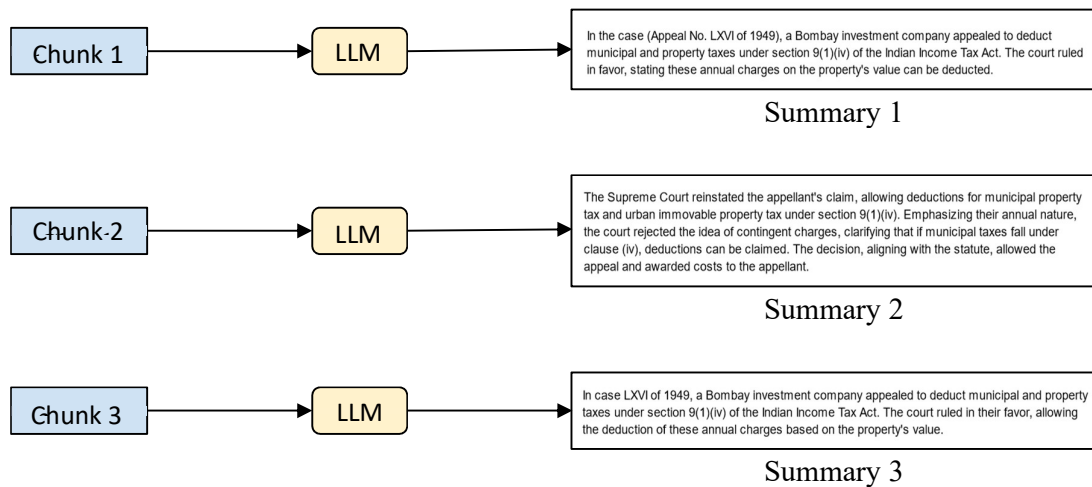


Figure 3.9. Summarizing chunks

4. ***Combining all the summaries into one concise summary:*** Combine all the obtained summaries into one to get a clear and concise summary of the legal document.

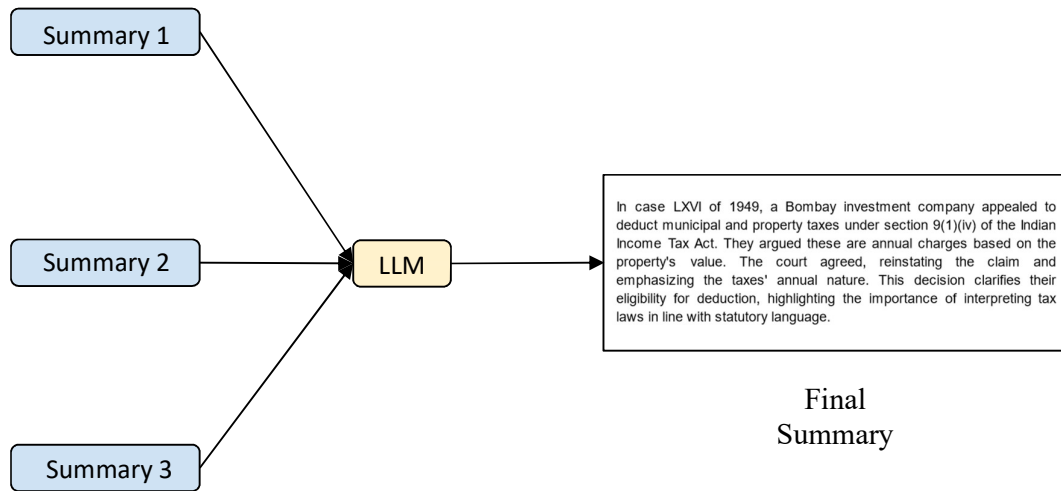


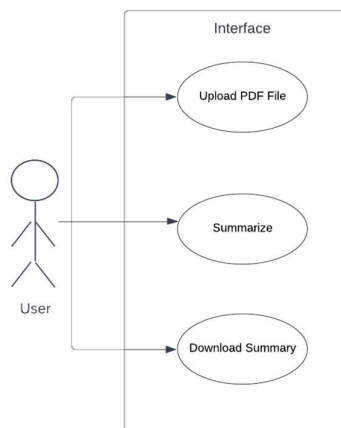
Figure 3.10. Combining the summaries

4. Design

Unified Modeling Language (UML) diagrams are visual representations used to model the structure and behavior of software systems. UML diagrams provide a standardized way to communicate system architecture, design, and functionality among stakeholders, including developers, designers, project managers, and clients. With a wide range of diagram types, including class diagrams, use case diagrams, sequence diagrams, activity diagrams, and more, UML offers a comprehensive toolkit for capturing different aspects of software systems. Each diagram type serves a specific purpose, allowing stakeholders to focus on different facets of the system, such as static structure, dynamic behavior, interactions, and processes. By using UML diagrams, stakeholders can effectively communicate, analyze, and understand complex software systems, aiding in requirements elicitation, system design, implementation, and maintenance phases of software development projects. UML diagrams serve as a common language for expressing system concepts and relationships, fostering collaboration, facilitating decision-making, and ultimately improving the quality and success of software projects.

4.1. Use case diagram

A use case diagram is a graphical representation in Unified Modelling Language (UML) that illustrates the functionalities or actions a system offers to its users (actors). Acting as a high-level overview of system behaviour from a user's perspective, it helps in understanding the system's requirements and interactions.

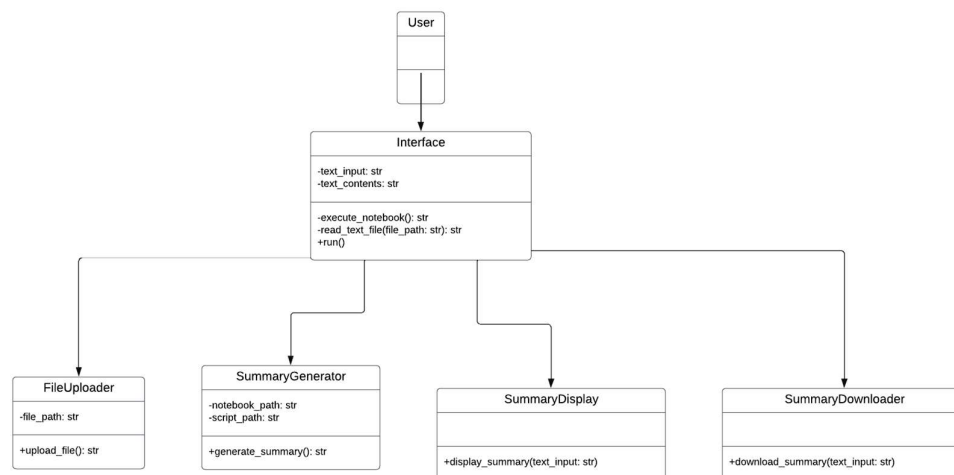


4.1. Use case diagram

In the context of a Streamlit application for summarizing PDF documents, the use case diagram showcases key functionalities that enable users to interact with the system seamlessly. Firstly, the "Upload PDF File" use case signifies the user's ability to upload a PDF document into the application. Following this, the "Summarize" use case represents the action of generating a summary of the uploaded document. Once the summary is generated, the "Display Summary" use case depicts how the system presents the summary to the user, facilitating easy comprehension. Lastly, the "Download Summary" use case allows the user to save the generated summary for future reference or distribution. Together, these use cases delineate the core interactions between the user and the Streamlit application, outlining the primary functionalities and user-driven actions that define the system's behavior.

4.2. Class diagram

A class diagram is a fundamental type of Unified Modelling Language (UML) diagram used in software engineering to visualize the structure of a system by representing its classes, attributes, operations, and relationships. Each class is depicted as a box with three compartments: the top compartment contains the class name, the middle compartment lists the attributes or properties of the class, and the bottom compartment displays the operations or methods that the class can perform. The relationships between classes, such as associations, dependencies, generalizations, and aggregations, are represented by lines connecting the classes. These relationships provide insight into how classes collaborate and interact within the system.

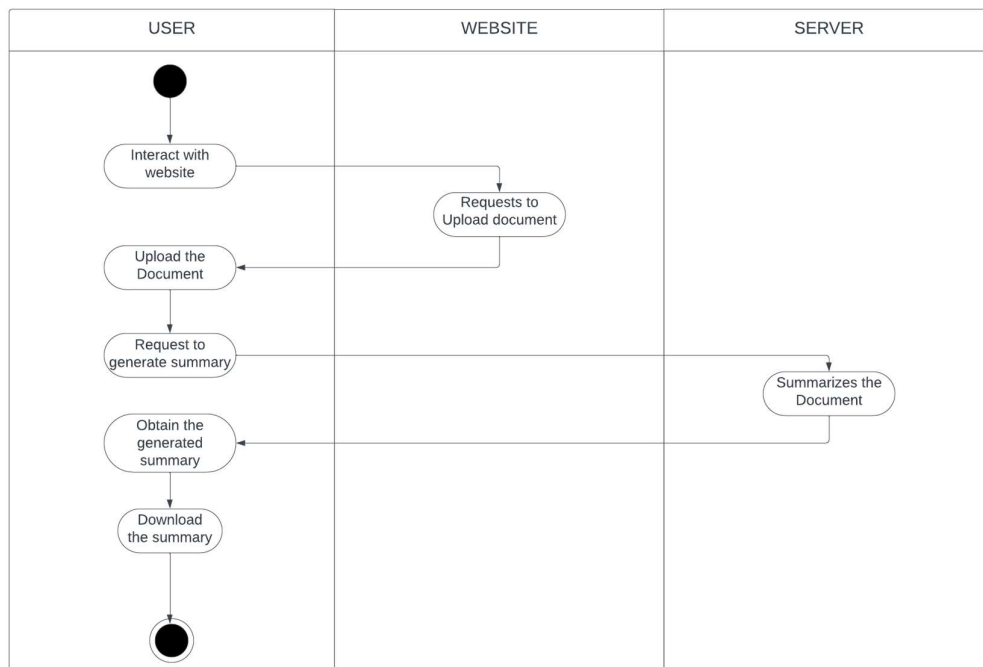


4.2. Class diagram

In the provided class diagram, several classes are depicted, each representing a distinct component of the system. For example, the StreamlitApp class encapsulates the main functionalities of the Streamlit application, including text input/output and notebook execution. Other classes such as FileUploader, SummaryGenerator, SummaryDisplay, and SummaryDownloader represent specific functionalities within the system, such as uploading files, generating summaries, displaying summaries, and downloading summaries, respectively. The arrows between classes indicate relationships, demonstrating how these components interact with each other. Overall, the class diagram provides a comprehensive overview of the system's structure, helping stakeholders understand its components, functionalities, and relationships, thereby facilitating effective communication, design, and development of the software system.

4.3. Activity Diagram

An activity diagram is a type of Unified Modelling Language (UML) diagram used to visually represent the flow of activities or actions within a system. It provides a structured way to illustrate the sequence of steps and decision points involved in a process.

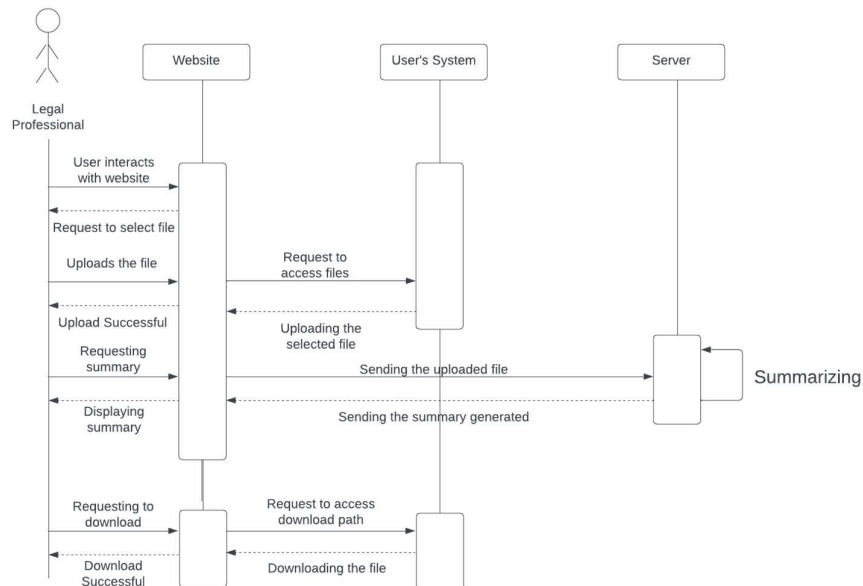


4.3. Activity diagram

In the given activity diagram, the process flow of a Streamlit application for summarizing PDF files is depicted. The diagram begins with the user uploading a PDF file. The application then checks if the file has been successfully uploaded. If so, it proceeds to read the file path and verifies if the file exists. Upon confirmation, the application executes a notebook to generate a summary of the document, followed by displaying the generated summary to the user. Subsequently, the application checks if the user intends to download the summary. If the user desires to do so, the summary is downloaded, and the process ends. However, if the file upload fails or the file does not exist, appropriate error messages are displayed, and the process is terminated. This activity diagram effectively outlines the sequential steps and decision points involved in the application's workflow, guiding users through the process of uploading, summarizing, displaying, and potentially downloading the generated summary.

4.4. Sequence diagram

A sequence diagram is a type of Unified Modelling Language (UML) diagram used to visualize interactions between objects or components in a system over time. It illustrates the flow of messages or method calls between these objects, providing a detailed view of the sequence of actions during the execution of a particular scenario.



4.4. Sequence diagram

In the provided sequence diagram, the interaction between the user and the Streamlit application for summarizing PDF files is depicted. The sequence starts with the user uploading a PDF file to the application. Upon receiving the upload request, the Streamlit application is activated, and it sequentially performs several actions, including reading the file path, executing a notebook to generate a summary, and displaying the generated summary. Finally, the application offers the option to download the summary, after which it sends the summary back to the user. Once the summary is displayed to the user, the Streamlit application is deactivated, indicating the end of the interaction sequence. This sequence diagram effectively captures the chronological flow of interactions between the user and the Streamlit application, illustrating how the system processes the user's actions to generate and present the summary.

5. Implementation

Program file LLTS.ipynb consists of the code for creating a summarizer, *interface.py* is the user interface.

Input: Legal Document

Output: Summary of the legal document

5.1. Functionality:

5.1.1. Loading and processing the legal document:

The process starts with loading and processing the legal document. It involves initializing a PyPDFLoader object named "loader" using a provided PDF file path. This loader is responsible for loading and processing the contents of the PDF file.

1. ***Splitting the document:*** Once the loader is initialized, it is used to load and split the PDF document into smaller "docs" or document chunks. These document chunks likely represent different sections or pages of the PDF file. This step is essential for further analysis or processing, as it breaks down the document into more manageable parts.
2. ***Returning the document chunks:*** Finally, the process returns the list of document chunks. These chunks are now available for further processing or analysis. This step completes the initial loading and processing phase, making the document content ready for subsequent operations.
3. ***Text Splitter:*** This is mentioned to overcome a token limit. It's a mechanism designed to break down the text into smaller chunks, each within the token limit. This is likely necessary because the subsequent processing involves a language model that has a token capacity, and the text needs to be divided accordingly to fit within that capacity.
4. ***Chunk overlap:*** The "chunk_overlap" parameter is mentioned, indicating that there's some allowance for overlap between the document chunks during the splitting process. This ensures that no information is lost as the document is divided into smaller parts. Overlapping chunks help maintain context and

coherence, especially when dealing with continuous text or sections that span across multiple chunks.

5.1.2. Combining documents chain:

This chain plays a crucial role in the document summarization process. It takes the individual legal document summaries, generated in the "Map" step, and combines them into a single, cohesive text string. By consolidating the summaries, it prepares the data for further processing in the "Reduce" step. The resulting combined document string is assigned the variable name "doc_summaries."

5.1.3. Reduce documents chain:

This chain represents the final phase of the summarization process. Its primary function is to take the combined document string from the `combine_documents_chain` and perform in-depth reduction and summarization. To address potential issues related to token limits (where documents may exceed a certain token count), this chain offers a clever solution. It can recursively collapse or compress lengthy documents into smaller, more manageable chunks. This ensures that the summarization process remains efficient and avoids token limit constraints. The maximum token limit for each chunk is set at 5,000 tokens, helping control the size of the summarization output.

5.1.4. Map-Reduce documents chain:

This chain follows the well-known MapReduce paradigm, a framework often used in distributed computing for processing and generating large datasets. In the "Map" step, it employs the `map_chain` to process each individual legal document. This results in initial document summaries. In the subsequent "Reduce" step, the chain uses the `reduce_documents_chain` to consolidate these initial summaries into a final, comprehensive document summary. The summarization result, representing the distilled insights from the legal documents, is stored in the variable named "docs" within the LLM chain.

5.2. Attributes:

i. API_KEY: It is a unique identifier or token that is used to authenticate and authorize access to OpenAI API.

ii. temperature: The temperature attribute controls the randomness of the generated text. A higher temperature value leads to more randomness and diversity in the generated responses, while a lower value results in more conservative and predictable responses. Here, it's set to 0, indicating that the responses will be deterministic rather than varied.

iii. max_tokens: This attribute specifies the maximum number of tokens (basic units of text) allowed in the generated response. Setting a limit helps manage the length of responses and prevents excessively long outputs. Here, it's set to 1000 tokens.

iv. openai_api_key: This attribute represents the API key required to authenticate and access the OpenAI API. The API key serves as a form of authentication and authorization to use OpenAI's services. It's typically obtained by registering with OpenAI and generating a unique key associated with the user's account. In this code, API_KEY is a placeholder for the actual API key value.

v. model: This attribute specifies the model to be used for generating responses. It determines the underlying architecture and parameters of the language model. Here, the model being used is 'ft:gpt-3.5-turbo-0613:personal::8f3ZRSq', which appears to be a fine-tuned version of the GPT-3.5 model. The specifics of the fine-tuning process and the particular characteristics of this model are indicated by the string.

vi. chunk_size: This attribute specifies the size of each chunk into which the text will be divided. Here, it's set to 1000, meaning that the text will be split into chunks, each containing a maximum of 1000 characters.

vii. chunk_overlap: This attribute controls the overlap between adjacent chunks. Overlapping ensures that no information is lost at the boundary of the chunks. A value

of 50 indicates that there will be an overlap of 50 characters between adjacent chunks.

viii. separator: This attribute specifies the separator to be used between chunks. In this case, it's set to '\n\n', which represents two newline characters. This separator will be inserted between each chunk, making it easier to distinguish between chunks when processing them separately.

ix. map_template: This is a multi-line string that serves as a template for generating prompts. It contains a placeholder {docs} which will be replaced with a list of documents when generating a prompt. The template also provides some context and instructions for the task to be performed, which involves summarizing key information from the provided documents.

x. reduce_template: This is another multi-line string serving as a template for generating prompts. It includes a placeholder {doc_summaries} which will be replaced with a set of document summaries when generating a prompt. The template provides instructions for the task to be performed, which involves distilling the provided summaries into a final consolidated summary using simple language, including all important sentences from the individual summaries, and presenting arguments with titles in bold.

xi. llm_chain: This parameter specifies an instance of an LLMChain called reduce_chain. This suggests that the StuffDocumentsChain will utilize the functionality provided by the reduce_chain instance, which was initialized earlier and is associated with a language model and a prompt template for summarizing documents.

xii. document_variable_name: This parameter defines the variable name to be used for the combined documents. It indicates that the documents being combined are stored in a variable named "doc_summaries". This likely refers to the document summaries generated earlier in the process.

xiii. return_intermediate_steps: This parameter determines whether intermediate steps of the process should be returned. In this case, it's set to False, suggesting that only the final result of the process (the comprehensive document summary) will be returned, without any intermediate steps.

5.3. Experimental Screenshot

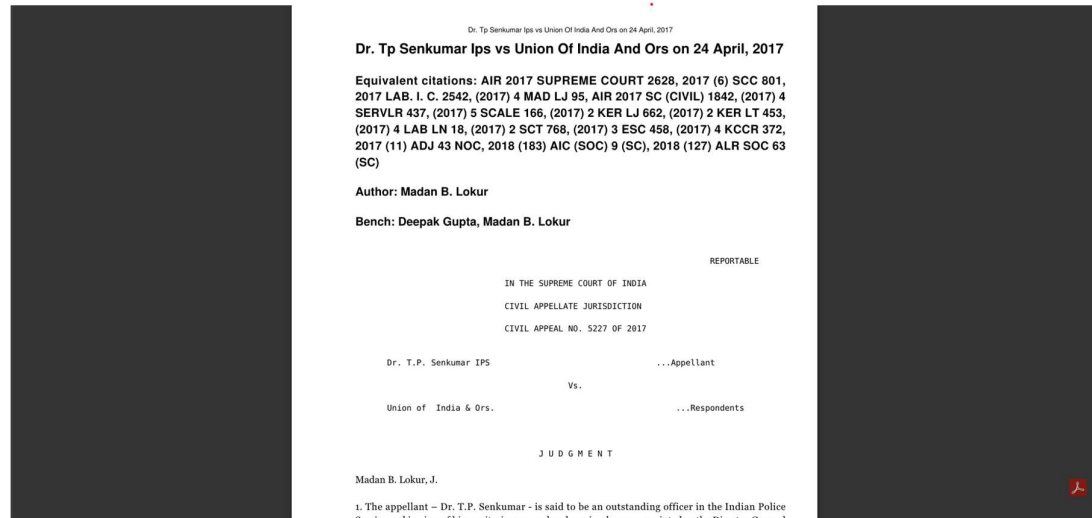


Figure 5.1. Legal Document

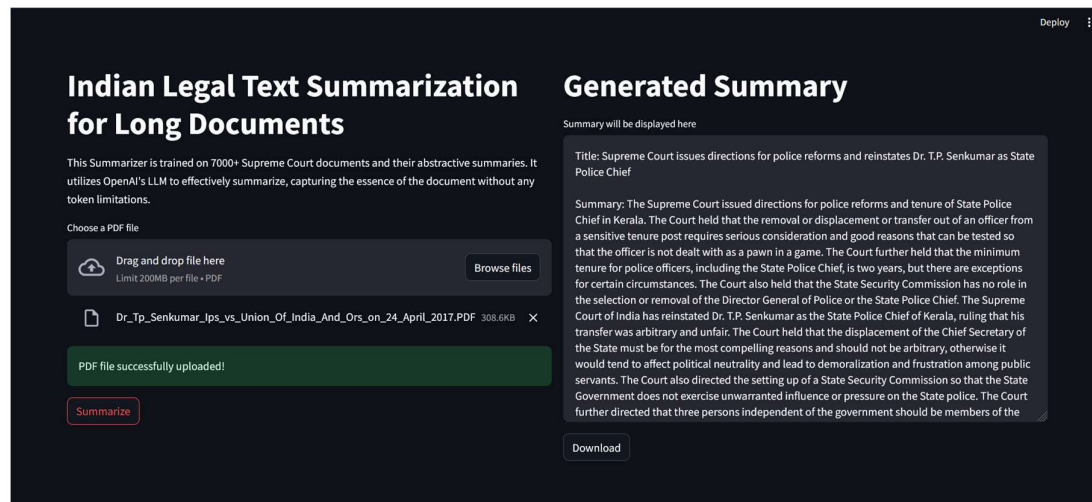
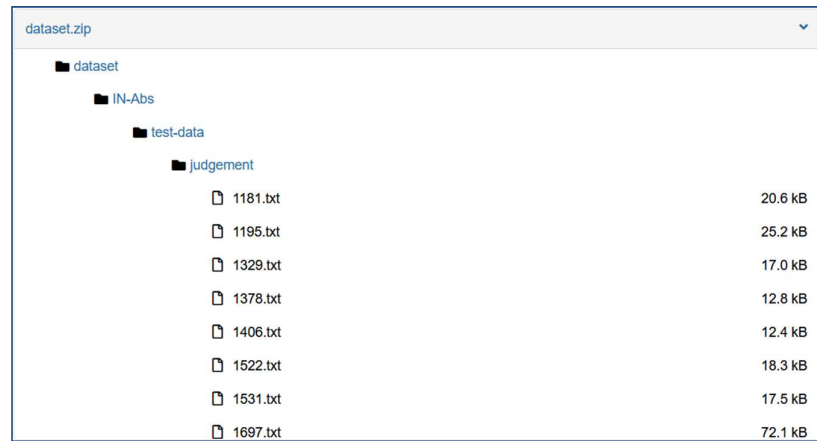


Figure 5.2. Output

5.4. Dataset

The repository under consideration is dedicated to facilitating legal document summarization, particularly focusing on Extractive and Abstractive methods along with their evaluation. It offers a comprehensive dataset tailored for this purpose, prominently featuring the IN-Abs dataset. This dataset comprises over 7000 Indian Supreme Court case documents paired with their respective 'abstractive' summaries. These documents and summaries are sourced from a credible legal database, specifically <http://www.liiofindia.org/in/cases/cen/INSC/>. This dataset serves as a valuable resource for researchers and practitioners in the field of legal document summarization, providing ample material for training and evaluating algorithms designed for both extractive and abstractive summarization techniques.



dataset.zip	
dataset	
IN-Abs	
test-data	
judgement	
1181.txt	20.6 kB
1195.txt	25.2 kB
1329.txt	17.0 kB
1378.txt	12.8 kB
1406.txt	12.4 kB
1522.txt	18.3 kB
1531.txt	17.5 kB
1697.txt	72.1 kB

Figure 5.3. Dataset

6. Experimental Setup

Used **OpenAI API Key**, **Jupyter Notebook**, **streamlit** to develop this Indian Legal text summarizer. API key will be used to access LLM and fine-tune LLM, Jupyter Notebook will be used to create a summarizer with map-reduce paradigm, and streamlit will be used to create interface for end user.

6.1. Obtain OpenAI API Key

To obtain an OpenAI API key, you need to follow these steps:

1. **Create an OpenAI Account:** If you haven't already, sign up for an account on the OpenAI website.
2. **Navigate to API Settings:** Log in to your OpenAI account and go to the API settings page. This is where you'll find your API key and manage your API usage.
3. **Generate an API Key:** If you haven't generated an API key yet, you'll need to create one. Click on the button or link to generate a new API key.
4. **Copy the API Key:** Once the API key is generated, you'll see it displayed on the screen. Copy this key to your clipboard.
5. **Store the API Key Securely:** It's essential to handle API keys securely. Avoid hardcoding API keys directly into your code, especially if you're sharing your code publicly. Instead, you can use environment variables or configuration files to store and access the API key.

6.2. Setup Jupyter Notebook:

To install and set up Jupyter Notebook, you can follow these steps:

1. **Install Python:** First, you need to have Python installed on your system. You can download and install Python from the official Python website: <https://www.python.org/>. Make sure to check the option to add Python to your system PATH during installation.

2. ***Install Jupyter Notebook***: Once Python is installed, you can install Jupyter Notebook using pip, which is the Python package manager. Open a terminal or command prompt and run the following command:

```
pip install jupyter
```

3. ***Launch Jupyter Notebook***: After the installation is complete, you can launch Jupyter Notebook by running the following command in your terminal or command prompt:

```
jupyter notebook
```

4. ***Accessing Jupyter Notebook***: Once you run the command, your default web browser should open, and you'll be directed to the Jupyter Notebook dashboard. If it doesn't open automatically, you can manually open your web browser and go to <http://localhost:8888/>. Here, you'll see a file browser where you can navigate your filesystem and create or open Jupyter Notebook files.

5. ***Creating a New Notebook***: To create a new notebook, click on the "New" button in the top right corner and select "Python 3" (or any other available kernel you want to use).

6. ***Using Jupyter Notebook***: You can now start using Jupyter Notebook. Each notebook consists of cells where you can write and execute Python code, Markdown for documentation, and more. You can execute a cell by pressing Shift+Enter or by clicking the "Run" button in the toolbar.

7. ***Saving and Closing***: Make sure to save your work regularly by clicking the "Save" button or using the keyboard shortcut Ctrl+S. To close Jupyter Notebook, you can simply close the browser tab or stop the Jupyter Notebook server by pressing Ctrl+C in the terminal or command prompt where it's running.

6.3. Setup Streamlit:

To install and set up Streamlit, a popular Python library for creating interactive web applications, follow these steps:

1. **Install Python:** Ensure you have Python installed on your system. You can download and install Python from the official Python website: <https://www.python.org/>. Make sure to check the option to add Python to your system PATH during installation.

2. **Install Streamlit:** You can install Streamlit using pip, the Python package manager. Open a terminal or command prompt and run the following command:

```
pip install streamlit
```

3. **Create a Streamlit Application:** Once Streamlit is installed, you can create a new Python script (`.py` file) to define your Streamlit application. For example, create a file named `app.py`.

4. **Write Streamlit Code:** Write your Streamlit application code in `app.py`. Streamlit provides a simple and intuitive API for creating web applications directly from Python scripts.

5. **Run Streamlit Application:** In your terminal or command prompt, navigate to the directory containing `app.py` and run the following command:

```
streamlit run app.py
```

6. **Access Your Streamlit App:** Once the Streamlit server starts, it will provide a local URL (usually <http://localhost:8501>) where you can access your Streamlit application in your web browser.

7. **Develop Your Application:** You can continue to develop your Streamlit application by modifying `app.py`. Streamlit provides numerous components and features for building interactive data-driven applications, including widgets, charts, layouts, and more. Refer to the Streamlit documentation for detailed information: <https://docs.streamlit.io/>

8. ***Deploy Your Application:*** Once you're satisfied with your Streamlit application, you can deploy it to various platforms, including Streamlit Sharing, Heroku, AWS, or any other hosting provider.

6.4. Libraries Used:

6.4.1. OpenAI

OpenAI Installed offers seamless access to OpenAI's advanced large language models like GPT (Generative Pre-trained Transformer) for various natural language processing tasks. It encompasses the installation of essential libraries and dependencies, acquiring an API key for authentication, integration of OpenAI's functionalities into applications or workflows, and leveraging the models for diverse tasks such as text generation, summarization, translation, and beyond. Additionally, it provides the flexibility of optional customization, enabling fine-tuning or adaptation of models to meet specific requirements, thus empowering developers and researchers with cutting-edge language processing capabilities. Whether it's crafting compelling narratives, extracting insights from large volumes of text, or enhancing communication across languages, OpenAI Installed serves as a robust toolkit for harnessing the power of artificial intelligence in the realm of language understanding and generation. With its user-friendly interface and powerful features, OpenAI Installed paves the way for innovative applications across various domains, revolutionizing how we interact with and derive value from textual data in today's digital age.

6.4.2. LangChain

LangChain stands as an indispensable tool in the realm of document management, playing a pivotal role in streamlining the intricate processes of document mapping, reduction, and combining with unparalleled efficiency. Its multifaceted functionality serves as a cornerstone for seamlessly integrating diverse documents, facilitating swift and precise mapping processes that ensure accuracy and coherence across various sources. Beyond mere integration, LangChain's prowess extends to the reduction of redundant information and the optimization of overall document structure, thereby enhancing readability and usability. Its adeptness in combining disparate documents further amplifies productivity by consolidating relevant data and eliminating

inconsistencies, leading to streamlined workflows and enhanced organizational efficiency. As organizations grapple with increasingly complex data landscapes, LangChain emerges as an essential component, offering a robust solution for managing the intricacies of document management with precision and effectiveness. Whether it's aligning disparate sources, optimizing content structures, or ensuring consistency across documents, LangChain stands as a beacon of efficiency, empowering organizations to navigate the challenges of document management with confidence and ease. In essence, LangChain transcends being just a tool; it becomes a strategic asset for organizations seeking to optimize their document management workflows and drive tangible results in today's fast-paced digital landscape.

6.4.3. Tiktoken

Tiktoken emerges as a pivotal tool in the landscape of language processing, specifically tailored to address the challenges posed by token count limitations within textual data. With a primary goal of optimizing the utilization of language models while circumventing token limit constraints, Tiktoken introduces an innovative solution that revolutionizes the efficiency of language processing tasks. At its core, Tiktoken operates by meticulously managing the distribution and allocation of tokens, strategically navigating the intricate balance between resource optimization and task completion. This intricate process ensures that users can seamlessly navigate through potential limitations, thereby enabling smooth and uninterrupted operations when interacting with language models. The significance of Tiktoken becomes apparent in scenarios where token count constraints pose barriers to effective language processing. Whether it's parsing through large volumes of text, generating summaries, or conducting sentiment analysis, the ability to efficiently manage token usage is paramount. Tiktoken rises to this challenge by providing a sophisticated framework that intelligently regulates token usage, allowing users to maximize the utility of language models without being hindered by arbitrary constraints. Moreover, Tiktoken's impact extends beyond mere efficiency gains; it fundamentally transforms the way users interact with language models, unlocking new possibilities for productivity and resource utilization. By seamlessly navigating token limit constraints, Tiktoken empowers users to leverage language models to their full potential, thereby facilitating more effective utilization of computational resources and bolstering overall productivity. In essence, Tiktoken serves as a catalyst for innovation in the field of

language processing, offering a vital toolset for overcoming token count limitations and optimizing the utilization of language models. Its nuanced approach to token management not only streamlines operations but also fosters a more seamless and productive workflow for users across various domains. As the demand for sophisticated language processing solutions continues to grow, Tiktoken stands at the forefront, driving advancements that redefine the boundaries of what's possible in the realm of textual data analysis and interpretation.

6.4.4. PyPdf

PyPDF stands out as a versatile Python library meticulously crafted to cater to the needs of individuals navigating the intricate landscape of legal documents and summaries. Its robust capabilities serve as a beacon of efficiency, empowering users to seamlessly access, analyze, and distill complex legal texts into concise summaries. The library's intuitive interface and powerful functionalities revolutionize the way users interact with legal documents, streamlining research processes and fostering enhanced comprehension. At the heart of PyPDF lies its ability to facilitate efficient navigation through legal texts. Users can effortlessly sift through extensive documents, leveraging PyPDF's advanced parsing algorithms to pinpoint relevant sections and extract key information. This capability not only saves time but also ensures that users can quickly access the data they need, minimizing the tedious manual effort traditionally associated with legal research. Furthermore, PyPDF empowers users to extract pertinent information from legal documents with remarkable precision. By leveraging its sophisticated extraction tools, users can identify crucial data points, such as case citations, statutes, and legal precedents, with ease. This feature not only accelerates the research process but also enhances the accuracy of analyses, enabling users to make well-informed decisions based on reliable information. Perhaps most notably, PyPDF excels in the generation of concise summaries, distilling complex legal texts into digestible insights. Leveraging its summarization capabilities, users can automatically generate succinct overviews of lengthy documents, providing valuable context and highlighting key findings. This functionality not only streamlines the consumption of legal information but also promotes greater understanding and clarity, empowering users to navigate complex legal landscapes with confidence. Overall, PyPDF emerges as an indispensable tool for individuals across various legal domains, offering a comprehensive suite of features designed to enhance efficiency and accuracy in legal

research endeavors. Whether parsing through statutes, analyzing case law, or preparing briefs, PyPDF's intuitive functionalities enable users to navigate the complexities of legal documents with ease, ultimately fostering greater productivity and insight. In a rapidly evolving legal landscape where access to timely and accurate information is paramount, PyPDF stands as a testament to the power of technology in empowering users to unlock the full potential of legal research.

The screenshot shows a JupyterLab interface with a notebook titled "Summarization using openai". The notebook contains three code cells:

```
[3]: from langchain.text_splitter import CharacterTextSplitter
text_splitter = CharacterTextSplitter.from_tiktoken_encoder(
    chunk_size=1000, chunk_overlap=50, separator='\n\n'
)
```

Splitting text by Character
The Text Splitter overcomes the token limit by breaking down the text into smaller chunks that are each within the token limit.

```
[4]: from langchain.document_loaders import PyPDFLoader
def file_loader(pdf_file_path):
    loader = PyPDFLoader(pdf_file_path)
    docs = loader.load()
    return docs
```

Loading legal document

```
[5]: from langchain.chains.mapreduce import MapReduceChain
from langchain.text_splitter import CharacterTextSplitter
from langchain.chains import ReduceDocumentsChain, MapReduceDocumentsChain
from langchain import PromptTemplate
from langchain.chains import LLMChain
from langchain.chains.combine_documents import StuffDocumentsChain
```

Map Reduce Prompt Template
Importing all the required for the implementation of LangChain MapReduce.

Template Definition
Two templates map_template and reduce_template are the structured prompts for instructing a language model on how to process and summarise sets of documents.

Figure. 6.1. Coding environment screenshot

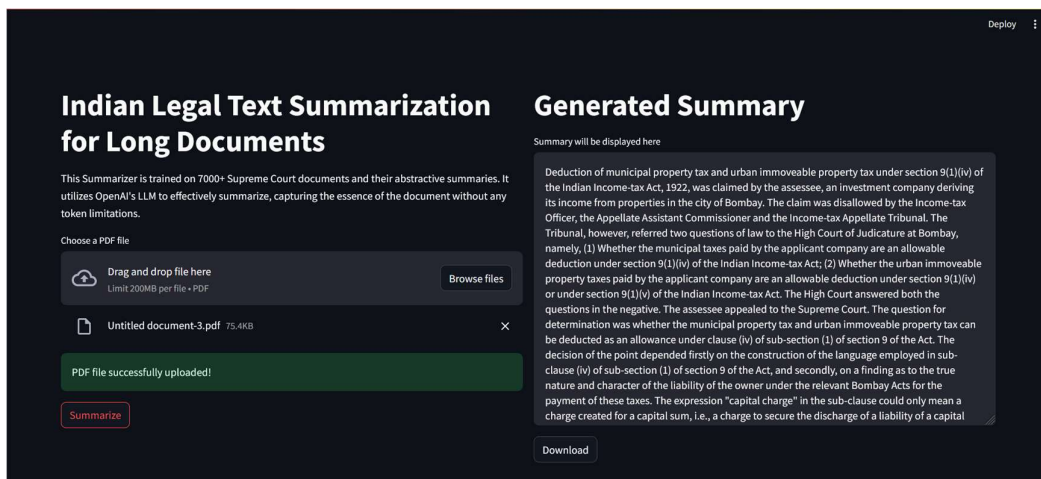


Figure. 6.2. User Interface

6.5. Parameters

The definition of ROUGE-N or ROUGE-L Metrics in terms of precision, recall, and F1 Scores parameters are elucidated as follows.

Precision denotes the fraction of the summary content that is pertinent to the original document. It is computed by dividing the count of relevant sentences in the summary by the total number of sentences in the summary, as illustrated in Equation (6.1).

$$\begin{aligned} & Precision_{ROUGE} \\ &= \frac{\text{Common } n - \text{grams in generated summary and reference summary}}{\text{Number of } n - \text{grams in generated summary}} \end{aligned} \quad (6.1)$$

Recall serves as a metric for assessing the efficacy of a summarization algorithm, indicating the proportion of crucial information in the original text that is captured in the algorithm-generated summary. The equation for recall is presented in Equation (6.2).

$$\begin{aligned} & Recall_{ROUGE} \\ &= \frac{\text{Common } n - \text{grams in generated summary and reference summary}}{\text{Number of } n - \text{grams in reference summary}} \end{aligned} \quad (6.2)$$

The **F1 score** represents the harmonic mean of precision and recall, amalgamating both measures into a single score that achieves a balance between precision and recall, as demonstrated in Equation (6.3).

$$F1\ score_{ROUGE-L} = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (6.3)$$

Here, 'n' denotes the length of the n-gram under consideration (e.g., n = 1, 2, 3). These formulas are applicable for assessing the quality of a summary generated by an algorithm by juxtaposing it with a reference summary.

7. Discussion of Results

For assessing the effectiveness of system-generated summaries, we employed the ROUGE metric, an acronym for Recall-Oriented Understudy for Gisting Evaluation. In this context, "Gisting" refers to the extraction of the primary point from the text [31]. ROUGE serves as an evaluation matrix that compares the generated summary with a reference summary, calculating the overlap between the two-concerning n-gram, word sequences, and word pairs. A higher ROUGE score signifies a superior alignment between the summary and the reference summary. ROUGE comprises five variants, including ROUGE-N, ROUGE-S, ROUGE-L, ROUGE-W, and ROUGE-SU [32]. In this study, we specifically employed ROUGE-N and ROUGE-L metrics to analyze various summarization models, where N denotes the length of the n-gram, encompassing ROUGE-1 (unigram), ROUGE-2 (bigram), ROUGE-3 (trigram), and so forth.

In this study, we conducted experiments with five distinct models, namely TextRank [33], Summarunner [34], Pointer-Generator [22], BERTSumABs [17], KESG [35], and our model to assess their performance on a dataset comprising 10 legal judgments against LLTS. Our evaluation entails the presentation of results using Precision, Recall, ROUGE-1, ROUGE-2, and ROUGE-L metrics. Significantly, we attribute equal importance to both precision and recall metrics to comprehensively gauge the model's overall performance. Consequently, we opt for the model exhibiting the highest F1 score, where an elevated F1 score signifies superior overall performance.

Table 7.1 presents the average precision scores for various summarization techniques applied to 10 distinct legal documents. Our observations indicate that LLTS excels, achieving a precision score of 0.892, surpassing other methods. The Pointer Generation summarization follows closely with a precision score of 0.714.

Table 7.1. Average precision score of different models

Models	Average Precision Score
TextRank	0.365
Summarunner	0.417
Pointer Generator	0.714
BERTSumABs	0.283
KESG	0.319
LLTS	0.892

The precision score percentages for all six models are illustrated in Figure 7.1.

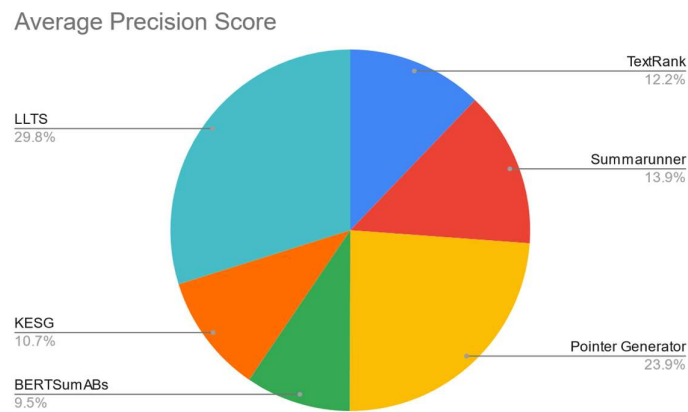


Figure 7.1. Precision score of models

Table 7.2. displays the Average Recall scores for various summarizer models applied to a given set of documents. Examining the table, it is evident that the LLTS exhibits the highest recall scores on average, closely followed by KESG. In contrast, Summarunner and Text rank demonstrate comparatively lower average recall scores.

Table 7.2. Average recall score of different models

Models	Average Recall Score
TextRank	0.12
Summarunner	0.29
Pointer Generator	0.35
BERTSumABs	0.40
KESG	0.48
LLTS	0.54

The recall score percentages for all six models are illustrated in Figure 7.2.

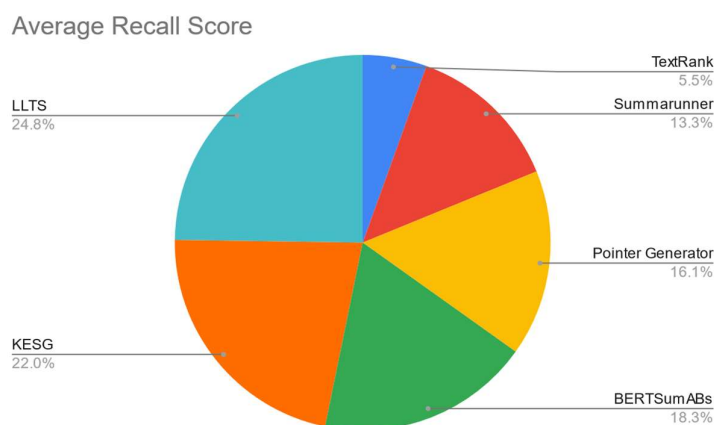


Fig. 7.2. Recall score of models

Table 7.3 displays the Average F1 scores of the summarizer models concerning ROUGE-1, ROUGE-2, and ROUGE-N for a given set of documents. According to the table, the LLTS achieves the highest ROUGE Scores, closely followed by Pointer Generation, whereas TextRank exhibits the lowest F1 scores on average.

Table 7.3. Average rouge score of different models

Models	ROUGE-1	ROUGE-2	ROUGE-N
TextRank	0.2057	0.0515	0.1353
Summarunner	0.2255	0.0625	0.1564
Pointer Generator	0.3837	0.2109	0.3026
BERTSumABs	0.4323	0.2694	0.3515
KESG	0.4351	0.2691	0.3521
LLTS	0.5399	0.3812	0.3621

The comparison of F1 scores for all six models are illustrated in Figure 7.3.

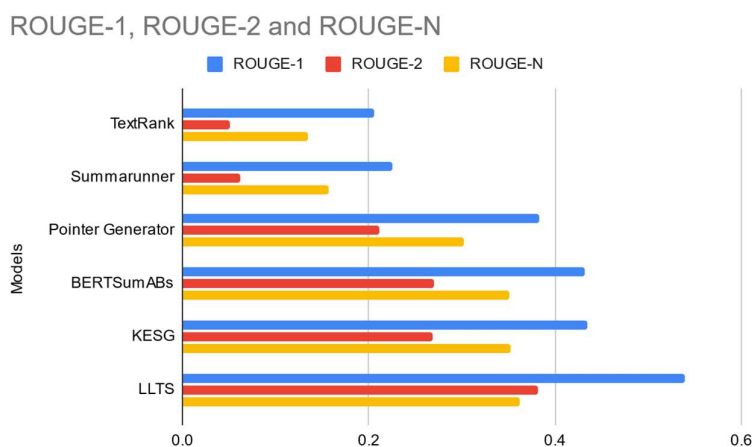


Figure 7.3. Comparison of the models

Upon thorough examination of the obtained outputs and results, it is evident that the LLTS provides concise and clear summaries while preserving all the legal nuances, regardless of the document size. In order to ascertain the optimal length for summarizing documents, an elaborate experiment was meticulously devised and executed. The experiment was predicated upon the utilization of three distinct files, each boasting a unique size: 308.6 KB, 712.4 KB, and 2497 KB. These files were meticulously chosen to represent a diverse spectrum of document sizes, thereby ensuring a comprehensive analysis.

The primary objective of the experiment was to explore how varying summary lengths, ranging from 10% to 40% of the original document length, would impact the efficacy and comprehensiveness of the summarization process. Each document was subjected to a series of summarizations, with the length of the summaries meticulously adjusted to adhere to the designated percentages of the original document length.

Table 7.4. Comparison of Summary Length to ROUGE-1 Score

S. No.	Original Document size in KB	Number of words in original document	Generated summary size in KB	Number of words in generated summary	ROUGE-1 Score
1	308.6	14395	9	1405	0.317
			18	2902	0.584
			28	4861	0.532
			35	5482	0.594
2	712.4	34120	21	3398	0.268
			43	6748	0.598
			63	10231	0.602
			79	12741	0.591
3	2497	119574	73	12046	0.219
			141	23953	0.473
			217	35452	0.495
			289	46892	0.490

After the meticulous process of summarization, the resulting summaries underwent comprehensive analysis through the lens of ROUGE scores. ROUGE, standing for Recall-Oriented Understudy for Gisting Evaluation, is a widely recognized metric used

to assess the quality of automatic summarization by comparing the generated summaries with reference summaries or human-produced summaries. This analysis aimed to provide quantitative insights into the effectiveness of each summary length in capturing the essence of the original documents. Upon conducting the ROUGE score analysis, intriguing patterns began to emerge, shedding light on the relationship between summary length and summarization efficacy. Notably, it was observed that summaries which encompassed 20% or more of the original document length exhibited notably higher ROUGE scores across various ROUGE metrics, indicating a superior ability to capture the gist and essence of the source material. Summaries crafted at lengths constituting 20% or more of the original document demonstrated a remarkable capacity to distill key information, ideas, and themes while retaining crucial nuances and contextual details. These summaries effectively encapsulated the core message of the source material, facilitating comprehension and retention among readers or recipients.

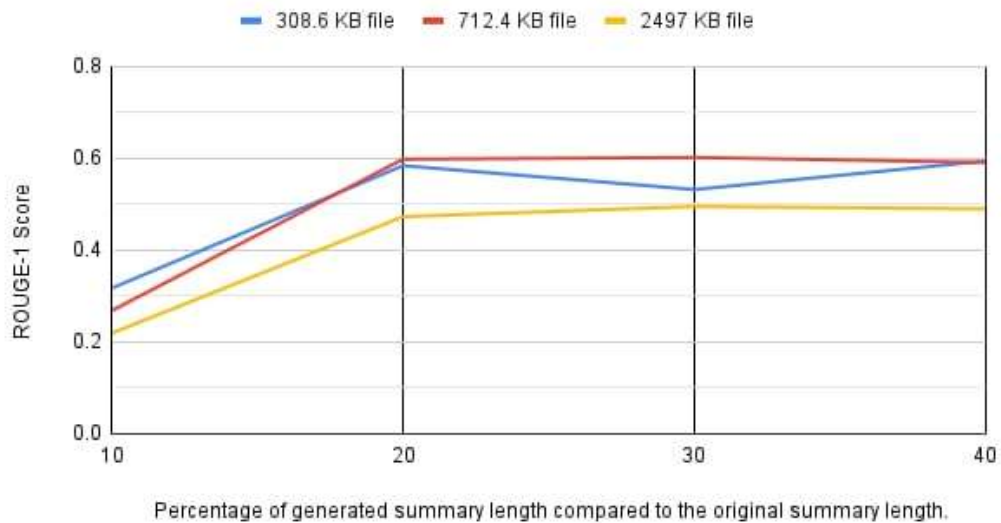


Figure 7.4. Summary Length v/s ROUGE-1 score

Furthermore, the observed correlation between summary length and ROUGE scores underscores the importance of striking a balance between brevity and comprehensiveness in the summarization process. While shorter summaries may offer succinctness and efficiency, they risk omitting crucial details and failing to convey the full scope of the original content. Conversely, longer summaries, though more expansive, may risk verbosity and dilution of the core message.

In conclusion, the ROUGE score analysis reaffirms the notion that the ideal summary length for capturing the gist effectively lies at 20% or more of the original document length. This finding serves as a cornerstone for future research and development efforts aimed at enhancing the efficacy and precision of automatic summarization algorithms, ultimately advancing the accessibility and usability of vast amounts of textual information.

8. Summary, Conclusion and Recommendation

The research findings highlight the efficacy of Large language model based Legal Text Summarization (LLTS) in accurately distilling the essence of entire legal documents, a task where other models often fall short. Unlike its counterparts, which struggle to produce concise summaries and grapple with the intricate nuances of legal language due to token limitations, LLTS offers a solution that effectively surmounts these obstacles. By leveraging the map-reduce paradigm and undergoing meticulous fine-tuning using a high-quality dataset, LLTS transcends token constraints and can generate summaries of any desired length.

LLTS's strength lies in its ability to provide comprehensive and nuanced summaries of legal documents, thereby enhancing the summarization process. By employing advanced techniques and harnessing the power of OpenAI's language model, LLTS captures the intricate details and essential elements of legal texts with remarkable accuracy. Its capability to overcome token limitations ensures that it can encapsulate the overall gist of a document without sacrificing depth or clarity.

The implementation of the map-reduce paradigm enables LLTS to efficiently process large volumes of data, breaking down the summarization task into manageable components and optimizing resource utilization. Additionally, the fine-tuning process enhances LLTS's understanding of legal terminology, syntax, and structure, enabling it to generate summaries that faithfully represent the original document.

Overall, LLTS represents a significant advancement in the field of legal document summarization, offering a solution that addresses the shortcomings of existing models and significantly improves the summarization process. Its ability to generate comprehensive and nuanced summaries underscores its effectiveness in facilitating efficient information retrieval and decision-making within legal contexts.

9. Future Enhancements

Future enhancements for LLTS could encompass several avenues. Firstly, there's a potential for domain-specific fine-tuning, honing the model's understanding within specific legal domains like contracts or intellectual property law. Additionally, extending LLTS to handle multi-document summarization could greatly benefit users dealing with related legal texts. Interactive summarization interfaces could facilitate iterative improvements based on user feedback. Implementing customizable summarization lengths would add flexibility to suit varied user requirements. Enhancing semantic understanding capabilities could lead to more nuanced and accurate summaries. Integration of legal citation recognition features could improve contextual accuracy. Continuous updates to adapt to regulatory changes would ensure the longevity and relevance of the summarization model. Multilingual support would extend LLTS's utility to diverse linguistic contexts. Developing specialized evaluation metrics tailored to legal text summarization could provide more precise assessment of LLTS performance. Lastly, integrating LLTS with existing legal research tools would streamline workflows for legal professionals and researchers.

10. Reference

- [1] S. Ghosh, M. Dutta and T. Das, “Indian Legal Text Summarization: A Text Normalization-based Approach,” 2022 IEEE 19th India Council International Conference (INDICON), Kochi, India, 2022, pp. 1-4, doi: 10.1109/INDICON56171.2022.10039891.
- [2] Mihalcea, R. and Tarau, P. (2004). “Textrank: Bringing order into texts”. In Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, pages 404–411. Association for Computational Linguistics.
- [3] Rush, A.M.; Chopra, S.; Weston, J. A neural attention model for abstractive sentence summarization. arXiv 2015, arXiv:1509.00685
- [4] Abhay Shukla, “Legal Case Document Summarization: Extractive and Abstractive Methods and their Evaluation”. Zenodo, Nov. 23, 2022. doi: 10.5281/zenodo.7151679.
- [5] <https://platform.openai.com/docs/guides/fine-tuning/>
- [6] M. C. Popescu, L. Grama and C. Rusu, "On the use of positive definite symmetric kernels for summary extraction," 2020 13th International Conference on Communications (COMM), Bucharest, Romania, 2020, pp. 335-340, doi: 10.1109/COMM48946.2020.9142041.
- [7] C. Popescu, L. Grama and C. Rusu, "Automatic Text Summarization by Mean-absolute Constrained Convex Optimization," 2018 41st International Conference on Telecommunications and Signal Processing (TSP), Athens, Greece, 2018, pp. 1-5, doi: 10.1109/TSP.2018.8441416.
- [8] A. Nenkova and K. McKeown, "Automatic summarization", *Foundations and Trends in Information Retrieval*, vol. 5, no. 2-3, pp. 103-233, 2011.
- [9] S. Polsley, P. Jhunjhunwala and R. Huang, *Casesummarizer: a system for automated summarization of legal texts*, pp. 258-262, 2016.
- [10] J. W. Yingjie and Ma, A comprehensive method for text summarization based on latent semantic analysis, Springer Berlin Heidelberg, pp. 394-401, 2013.

- [11] B. Samei, M. Estiagh, M. Eshtiagh, F. Keshtkar and S. Hashemi, *Multi-document summarization using graph-based iterative ranking algorithms and information theoretical distortion measures*, 2014.
- [12] A. Joshi, E. Fidalgo, E. Alegre and L. Fernández-Robles, "Summcode: An unsupervised framework for extractive text summarization based on deep auto-encoders", *Expert Systems with Applications*, vol. 129, pp. 200-215, 2019.
- [13] V. Parikh, V. Mathur, P. Mehta, N. Mittal and P. Majumder, *Lawsum: A weakly supervised approach for indian legal document summarization*, 10 2021.
- [14] Nguyen, Duy-Hung, Bao-Sinh Nguyen, Nguyen Viet Dung Nghiem, Dung Tien Le, Mim Amina Khatun, et al., "Robust Deep Reinforcement Learning for Extractive Legal Summarization", *International Conference on Neural Information Processing*, pp. 597-604, 2021.
- [15] Siyao Li, Deren Lei, Pengda Qin and William Yang Wang, *Deep reinforcement learning with distributional semantic rewards for abstractive summarization*, 2019.
- [16] K. Agrawal, "Legal Case Summarization: An Application for Text Summarization," *2020 International Conference on Computer Communication and Informatics (ICCCI)*, Coimbatore, India, 2020, pp. 1-6, doi: 10.1109/ICCCI48352.2020.9104093.
- [17] Y. Liu and M. Lapata, "Text Summarization with Pre Trained Encoders," in ArXiv, 2019, vol. abs/1908.08345,[Online]. Available: <https://api.semanticscholar.org/CorpusID:201304248>.
- [18] Milda Norkute, Nadja Herger, Leszek Michalak, Andrew Mulder, and Sally Gao. 2021. Towards Explainable AI: Assessing the Usefulness and Impact of Added Explainability Features in Legal Document Summarization. In Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems (CHI EA '21). Association for Computing Machinery, New York, NY, USA, Article 53, 1–7. <https://doi.org/10.1145/3411763.3443441>

- [19] K. Merchant and Y. Pande, "NLP Based Latent Semantic Analysis for Legal Text Summarization," 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Bangalore, India, 2018, pp. 1803-1807, doi: 10.1109/ICACCI.2018.8554831.
- [20] A. Farzindar, "Atefeh Farzindar and Guy Lapalme'LetSum an automatic Legal Text Summarizing system", *Legal Knowledge and Information Systems. Jurix 2004: The Seventeenth Annual Conference*, vol. 120, pp. 11-18, 2004, 2004
- [21] S. S. Megala, A. Kavitha and A. Marimuthu, "Feature extraction based legal document summarization", *International Journal of Advance Research in Computer Science and Management Studies*, vol. 2, no. 12, pp. 346-352, 2014.
- [22] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," *arXiv preprint arXiv:1704.04368*, 2017.
- [23] D. D. V. Feijo and V. P. Moreira, "Improving abstractive summarization of legal rulings through textual entailment" in *Artificial Intelligence and Law*, vol. 31, no. 1, pp. 91-113, 2023, Springer.
- [24] Ilias Chalkidis, Ion Androutsopoulos and Nikolaos Aletras, "Neural legal judgment prediction in English", *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4317-4323, July 2019.
- [25] A. Sleimi, N. Sannier, M. Sabetzadeh, L. Briand and J. Dann, "Automated Extraction of Semantic Legal Metadata using Natural Language Processing," 2018 IEEE 26th International Requirements Engineering Conference (RE), Banff, AB, Canada, 2018, pp. 124-135, doi: 10.1109/RE.2018.00022.
- [26] C. Prasad, J. S. Kallimani, D. Harekal and N. Sharma, "Automatic Text Summarization Model using Seq2Seq Technique," 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 2020, pp. 599-604, doi: 10.1109/I-SMAC49090.2020.9243572.
- [27] Hachey, B., Grover, C. Extractive summarisation of legal texts. *Artif Intell Law* 14, 305–345 (2006). <https://doi.org/10.1007/s10506-007-9039-z>

- [28] Farzindar Atefeh and Lapalme Guy, Legal text summarization by exploration of the thematic structures and argumentative roles, 2011
- [29] V. Ravi Kumar and K. Raghuveer, Legal Document Summarization using Latent Dirichlet Allocation, 2012, <https://api.semanticscholar.org/CorpusID:56116266>
- [30] M. Saravanan and B. Ravindran, "Identification of rhetorical roles for segmentation and summarization of a legal judgment", *Artif. Intell. Law*, vol. 18, no. 1, pp. 45-76, March 2010
- [31] Ng, J.-P. and Abrecht, V. (2015). Better summarization evaluation with word embeddings for rouge. arXiv preprint arXiv:1508.06034.
- [32] Kanapala A, Pal S, Pamula R. Text summarization from legal documents: a survey. *Artificial Intelligence Review*. 2019; 51:371–402.
- [33] Mihalcea, R.; Tarau, P. Textrank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, Barcelona, Spain, 25–29 October 2004; pp. 404–411
- [34] Nallapati, R.; Zhai, F.; Zhou, B. Summarunner: A recurrent neural network-based sequence model for extractive summarization of documents. In *Proceedings of the Thirty-first AAAI Conference on Artificial Intelligence*, San Francisco, CA, USA, 4–9 February 2017.
- [35] Deng, Z.; Ma, F.; Lan, R.; Huang, W.; Luo, X. A two-stage Chinese text summarization algorithm using keyword information and adversarial learning. *Neurocomputing* 2021, 425, 117–126.
- [36] N. Begum and A. Goyal, "Analysis of Legal Case Document Automated Summarizer," 2021 6th International Conference on Signal Processing, Computing and Control (ISPCC), Solan, India, 2021, pp. 533-538, doi: 10.1109/ISPCC53510.2021.9609442.
- [37] Moro, G., Piscaglia, N., Ragazzi, L. et. al. Multi-language transfer learning for low-resource legal case summarization. *Artif IntellLaw* (2023). <https://doi.org/10.1007/s10506-023-09373-8>.

- [38] Kore, Rahul & Ray, Prachi & Lade, Priyanka & Nerurkar, Amit. (2020). Legal Document Summarization Using NLP and ML Techniques. International Journal of Engineering and Computer Science. 9. 25039-25046. 10.18535/ijecs/v9i05.4488.
- [39] Blei, David & Ng, Andrew & Jordan, Michael. (2001). Latent Dirichlet Allocation. The Journal of Machine Learning Research. 3. 601-608.
- [40] M. Saravanan, B. Ravindran, and S. Raman. 2008. Automatic Identification of Rhetorical Roles using Conditional Random Fields for Legal Document Summarization. In Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-I.
- [41] Charles Sutton; Andrew McCallum, An Introduction to Conditional Random Fields, now, 2012, doi: 10.1561/22000000013.
- [42] G. Swain, Object-Oriented Analysis and Design Through Unified Modelling Language. Laxmi Publication, Ltd., 2010.
- [43] G. Booch, D. L. Bryan, and C. G. Peterson, Software engineering with Ada. Redwood city, Calif.: Benjamin/Cummings, 1994.