Large Language Model based Legal Text Summarization (LLTS)

MEGHA PARATE
Dept. Computer Science and Engineering
Anurag University
Hyderabad, Telangana, India.

***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. In this paper, a new dataset, recently developed and trained on Indian 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.**

***Index Terms*—GPT-3.5-turbo, Map-reduce, OpenAI**

I. INTRODUCTION

In the Indian judicial system, legal proceedings often extend over a prolonged period, and there are currently over 4 crore [1] pending cases awaiting resolution. The people involved in the legal process, like lawyers and judges, find it tedious and time-consuming to manually summarize lots of documents related to these cases. The situation highlights the importance of finding better and faster ways to handle information in the legal system, especially considering the large number of cases that are still pending.

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. Extractive summarization involves selecting and combining existing sentences or phrases directly from the source text to create a summary. In contrast, abstractive summarization generates a condensed version by rephrasing and paraphrasing the content. The choice between extractive and abstractive summarization depends on the specific requirements and goals of the task, each having its unique advantages and challenges.

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.

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 dictionary order to form generalized new sentences. Although their method was flexible to the generation method of natural summarization, it had some issues, such as factual bias and textual repetition.

The study concentrates on improving the summarization process through the utilization of an innovative dataset [5], consisting of 7000 judgments from the supreme court, accompanied by abstractive summaries created by legal experts. The summaries were generated using OpenAI's GPT-3.5-turbo model, fine-tuned with this specialized dataset [5]. 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.

To address the challenge of token limitations that could potentially restrict the length of summaries and obstruct comprehensive information inclusion, a map-reduce framework was implemented. The framework allowed overcoming token constraints and ensured that the generated summaries could accommodate all essential details from the legal documents.

II. LITERATURE REVIEW

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.'s [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. 1, assessing the relationships between sentences. The authors 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.

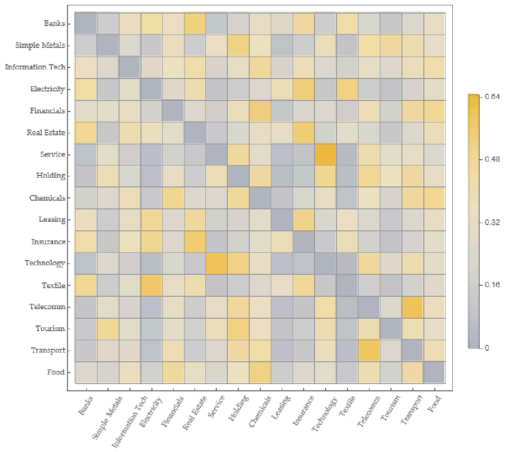

Figure. 1 Kernel Similarity Matrix

C. Popescu et al.'s [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. 's [12] approach involves several preprocessing 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.'s [13] approach involves preprocessing 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.'s [17] Summcoder 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. 's [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. 's [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 preprocessing 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 to compare similarity and the Jensen-Shannon Distance to quantify information difference between the summary and the original document.

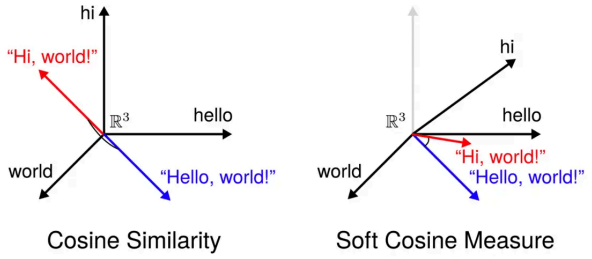

Figure. 2 Cosine Similarity

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.

III. PROPOSED METHOD

The proposed methodology has demonstrated a significant enhancement in ROUGE score, indicating that the generated summary effectively captures the essence of the entire document. Additionally, it has overcome the limitation of input document length by employing the 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.

A. 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. The fine-tuning process encompasses the following stages:

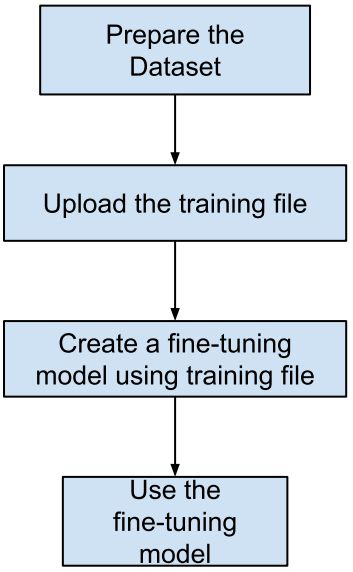

Figure. 3 Overview of fine-tuning procedure

1) Prepare and upload training data:

The dataset [4] comprises over 7000 judgments from the Indian Supreme Court along with their corresponding summaries. It is necessary to organize the dataset in a conversational format, similar to the structure used by the Chat Completions API. This involves presenting the data as a series of messages, where each message specifies a role and contains the actual message content. To achieve this, the dataset needs to be converted into a JSON Lines file format as shown in figure 4.

Consider an example representing Chat completion API format:

{ "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.

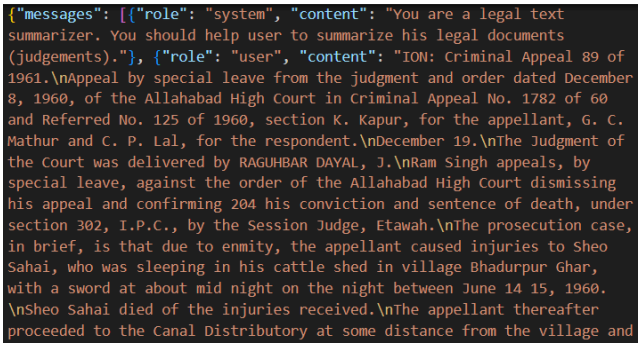

Figure. 4 JSON Lines dataset

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."

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.

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 12: 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 13: 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 14: Retrieve the fine-tuned model status using fine_tuning.job.retrieve() method under OpenAI. The model name will be generated.

B. Generating summaries using fine-tuned model:

In our legal document summarization process, we follow the Map-Reduce paradigm, 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.

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.

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) 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.

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.

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.

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.

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.

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.'

IV. Results And Discussion

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. The definition of ROUGE-N or ROUGE-L Metrics in terms of precision, recall, and F1 Scores parameters is 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 (1).

$$Precision_{ROUGE-L} = \frac{Common\ n-grams\ in\ generated\ summary\ and\ reference\ summary}{Number\ of\ \ n-grams\ in\ generated\ summary} \quad (1)$$

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 (2).

$$Recall_{ROUGE-L} = \frac{Common\ n-grams\ in\ generated\ summary\ and\ reference\ summary}{Number\ of\ \ n-grams\ in\ reference\ summary} \quad (2)$$

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 (3).

$$F1\ score_{ROUGE-L} = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (3)$$

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.

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 I 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 I

AVERAGE PRECISION SCORES OF DIFFERENT MODEL

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 Fig. 2.

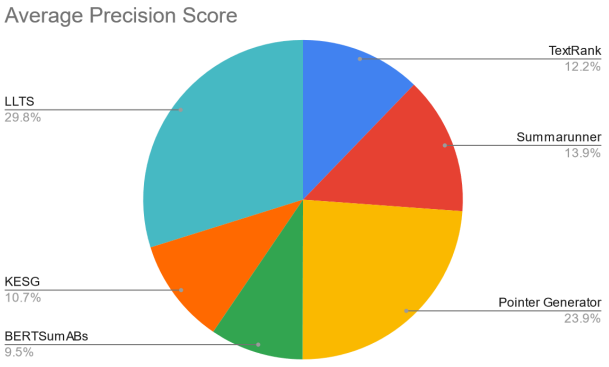

Fig. 2. Precision score of models

Table II 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 II

AVERAGE RECALL SCORES OF DIFFERENT MODEL

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 Fig. 3.

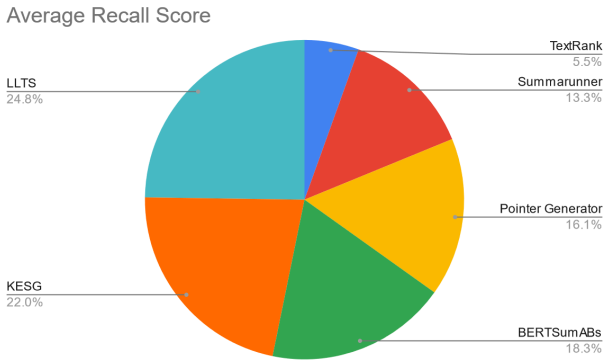

Fig. 3. Recall score of models

Table 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 III

AVERAGE ROUGE SCORES OF DIFFERENT MODEL

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 Fig. 4.

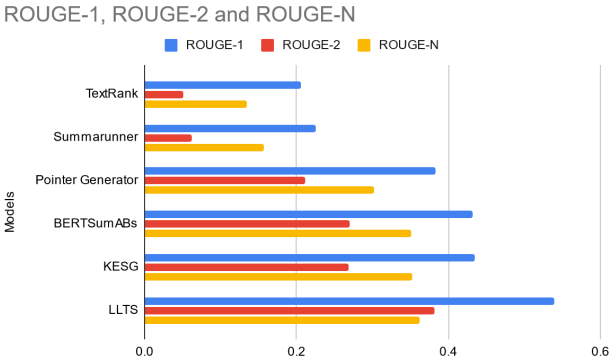

Fig. 3. Comparison of the models

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.

V. CONCLUSION

The research has demonstrated that the LLTS for legal text summarization stands out by accurately capturing the essence of entire legal documents. Unlike other models with limitations in generating concise summaries and struggling to grasp legal nuances, the proposed approach successfully overcomes these challenges. Through the implementation of the map-reduce paradigm and meticulous fine-tuning of OpenAI's language model with a high-quality dataset, LLTS excels in providing comprehensive and nuanced legal document summaries, showcasing its effectiveness in improving the summarization process.

VI. REFERENCES

[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] Munot, Nikita & Govilkar, Sharvari. (2014). Comparative study of text Summarization Methods. International Journal of Computer Applications. 102. 33-37 . 100.5120/17870-8810.

[3] 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.

[4] Rush, A.M.; Chopra, S.; Weston, J. A neural attention model for abstractive sentence summarization. arXiv 2015, arXiv:1509.00685

[5] Abhay Shukla, “Legal Case Document Summarization: Extractive and Abstractive Methods and their Evaluation”. Zenodo, Nov. 23, 2022. doi: 10.5281/zenodo.7151679.

[6] https://platform.openai.com/docs/guides/fine-tuning/

[7] 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.

[8] Sébastien Bubeck, Convex Optimization: Algorithms and Complexity , now, 2015.

[9] 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.

[10] "TF–IDF," Encyclopedia of Machine Learning, C. Sammut and G.I. Webb, Eds. Springer, Boston, MA, 2011, pp. 1–5. https://doi.org/10.1007/978-0-387-30164-8_832.

[11] A. Nenkova and K. McKeown, "Automatic summarization", *Foundations and Trends in Information Retrieval*, vol. 5, no. 2-3, pp. 103-233, 2011.

[12] S. Polsley, P. Jhunjhunwala and R. Huang, *Casesummarizer: a system for automated summarization of legal texts*, pp. 258-262, 2016.

[13] J. W. Yingjie and Ma, A comprehensive method for text summarization based on latent semantic analysis, Springer Berlin Heidelberg, pp. 394-401, 2013.

[14] V. Klema and A. Laub, "The singular value decomposition: Its computation and some applications," in IEEE Transactions on Automatic Control, vol. 25, no. 2, pp. 164-176, April 1980, doi: 10.1109/TAC.1980.1102314.

[15] 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.

[16] J.M. Joyce, "Kullback-Leibler Divergence," in M. Lovric (ed.), International Encyclopedia of Statistical Science. Springer, Berlin, Heidelberg, 2011, pp. 1–3. https://doi.org/10.1007/978-3-642-04898-2_327.

[17] A. Joshi, E. Fidalgo, E. Alegre and L. Fernández-Robles, "Summcoder: An unsupervised framework for extractive text summarization based on deep auto-encoders", *Expert Systems with Applications*, vol. 129, pp. 200-215, 2019.

[18] V. Parikh, V. Mathur, P. Mehta, N. Mittal and P. Majumder, *Lawsum: A weakly supervised approach for indian legal document summarization*, 10 2021.

[19] 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.

[20] Khan, M., Wang, H., Riaz, A. *et al.* Bidirectional LSTM-RNN-based hybrid deep learning frameworks for univariate time series classification. *J Supercomput* **77**, 7021–7045 (2021). https://doi.org/10.1007/s11227-020-03560-z

[21] Siyao Li, Deren Lei, Pengda Qin and William Yang Wang, *Deep reinforcement learning with distributional semantic rewards for abstractive summarization*, 2019.

[22] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. 2015. From word embeddings to document distances. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15). JMLR.org, 957–966.

[23] 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.

[24] 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.

[25] 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

[26] 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.

[27] P. Kherwa and P. Bansal, "Latent Semantic Analysis: An Approach to Understand Semantic of Text," 2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC), Mysore, India, 2017, pp. 870-874, doi: 10.1109/CTCEEC.2017.8455018.

[28] 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

[29] 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.

[30] B. Fuglede and F. Topsoe, "Jensen-Shannon divergence and Hilbert space embedding," International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings., Chicago, IL, USA, 2004, pp. 31-, doi: 10.1109/ISIT.2004.1365067.

[31] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," *arXiv preprint arXiv:1704.04368*, 2017.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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

[37] Farzindar Atefeh and Lapalme Guy, Legal text summarization by exploration of the thematic structures and argumentative roles, 2011

[38] V. Ravi Kumar and K. Raghuveer, Legal Document Summarization using Latent Dirichlet Allocation, 2012, https://api.semanticscholar.org/CorpusID:56116266

[39] Jelodar, H., Wang, Y., Yuan, C. *et al.* Latent Dirichlet allocation (LDA) and topic modeling: models, applications, a survey. *Multimed Tools Appl* 78, 15169–15211 (2019). https://doi.org/10.1007/s11042-018-6894-4

[40] 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

[41] Charles Sutton; Andrew McCallum, An Introduction to Conditional Random Fields , now, 2012.

[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.