# Automatic Text Summarizer Using LLM

**Meena Rao[1] and Parul Chaudhary[2]***

[1] Associate Professor, Maharaja Surajmal Institute of Technology, Janakpuri, New Delhi
[2] Assistant Professor, Maharaja Surajmal Institute of Technology, Janakpuri, New Delhi
*Corresponding Author

***Abstract***:
*This paper explores the field of automatic text summarization, addressing the challenge of efficiently extracting essential information from lengthy documents. Leveraging recent advancements in trainable Machine Learning algorithms, this paper proposes a novel summarization method that utilizes a comprehensive set of features derived directly from the original text. These features include both statistical metrics that indicate the frequency of key elements and linguistic attributes related to the text's argumentative structure.*

*The proposed summarization approach is evaluated through extensive experiments across various text databases, providing detailed computational results that highlight its effectiveness. Comparative analyses against established baseline methods demonstrate that the proposed technique outperforms others in generating concise and informative summaries.*

*Additionally, the paper emphasizes the growing significance of text summarization in light of the overwhelming increase in textual data on digital platforms. As the amount of information continues to expand, the demand for efficient summarization techniques becomes ever more critical, enabling quicker understanding and knowledge extraction.*

*Alongside the innovative summarization methodology, we offer insights into future research directions in this evolving field. By identifying emerging trends, challenges, and potential research avenues, we aim to stimulate further advancements in automatic text summarization technologies, ultimately improving information accessibility and knowledge dissemination in the digital age.*

**Keywords: Blockchain, IoT, cybersecurity, ISO27001, Smart contracts**

## I. Introduction:

The utilization of information has become costly and time-consuming due to the vast amounts of data generated at once, often containing irrelevant content or noise. Text summarization serves as a method to condense this data. While manual summarization is effective in preserving the text's meaning, it is also labour-intensive. An alternative is automatic text summarization (ATS), where various algorithms are programmed into computers to generate summaries.

Large language models (LLMs) are trained on extensive text data using deep learning techniques, enabling them to perform a wide range of natural language processing (NLP) tasks. These tasks can be broadly categorized into seven main areas: classification, clustering, extraction, generation, rewriting, search, and summarization.

Text summarization provides a concise and accurate overview of lengthy documents by focusing on essential details while maintaining overall context [1]. In NLP, automatic text summarization involves evaluating, understanding, and extracting information from human language. Today, professionals from students to researchers and business analysts frequently

work with numerous documents, often finding it challenging to locate relevant information. Here, ATS proves to be incredibly beneficial. Its primary aim is to create a compact and compelling summary that retains critical information.

As online information continues to grow, text summarization is increasingly important. Some of the earliest influential works in this area date back to 1958, when researchers suggested that word frequency could be a statistical measure in the summarization process, a concept still applicable in certain methods. For instance, with news articles, a user does not need to read through multiple pages; a brief summary often suffices. This has led to the development of applications that sift through hundreds of articles to provide personalized summary feeds. Similarly, social media platforms can analyze thousands of posts on a topic, identify overlapping content, and summarize it effectively.

Text summarization can also assist in directly answering user queries in search results, a function increasingly utilized by search engines. As the volume of shared information rises, the relevance of text summarization grows. There are two primary categories of summarization: extractive and abstractive. As the names suggest, extractive summarization focuses on identifying and selecting key sentences from the source text, while abstractive summarization involves understanding the main ideas and generating a new, concise summary.

The extractive method may lose some meaning due to the disconnection between selected sentences, whereas the abstractive method requires significant training to avoid grammatical and semantic errors when rewriting sentences. Abstractive summarization is language-dependent, whereas extractive summarization can be applied to various languages as the core concept remains consistent [2][3].

The two main types of summarizations are:

1. **Extractive Summarization:**
   This approach identifies and extracts key phrases or sentences from the source text to form a summary.
2. **Abstractive Summarization:**
   This method involves comprehending the main ideas in the source text and creating a new summary that conveys those ideas in a fresh, condensed manner.

**Need and Applications:**

**Here are the applications of the ATS domain:**

1. Book and Novel Summarization: ATS is primarily employed to condense lengthy documents such as books and novels, as shorter texts are generally not suitable for summarization. Long documents provide the necessary context for effective summarization.
2. Social Media and Tweet Summarization: With millions of messages generated daily on platforms like Facebook and Twitter, ATS can efficiently summarize important posts, tapping into this rich source of information.

3. Sentiment Analysis (SA): This process involves evaluating people's opinions and feelings about various events and situations. Using fuzzy logic, SA categorizes sentiments from product reviews as "Positive" or "Negative." ATS aids market analysts in summarizing the collective sentiments of large groups.
4. News Summarization: ATS is useful for condensing news articles from various outlets, such as CNN. It identifies the key points of a story, often serving as the headline.
5. Email Summarization: Given the unstructured nature of emails, ATS typically extracts noun phrases and generates summaries using linguistic techniques and machine learning algorithms.
6. Legal Document Summarization: ATS finds relevant precedents related to legal inquiries and rhetorical functions to summarize legal judgments. A hybrid approach utilizes methods like keyword extraction, critical phrase matching, and case-based analysis.
7. Biomedical Document Summarization: ATS employs genetic clustering and connectivity information within a graph-based summarization framework. Genetic clustering identifies themes in biomedical documents, while connectivity data illustrates the relative importance of the research.
8. Scientific Paper Summarization: Scientific papers are well-structured texts that present diverse viewpoints on similar topics. Critical information is often found in tables and figures rather than in the main text. A multi-document ATS framework integrates citation analysis and summarization techniques to generate comprehensive surveys of scientific literature.

## II. Related Work-

The text summarization techniques have gone through a pleothora of work and that includes analysis using Deep Learning (DL) and Machine Learning (ML) approaches. Mainly the works have focussed on statistical as well as linguistic data. In literature the work based on extractive text summarization identifies important features using certain primary words or keywords from the original article [4]. An important work presented in literature in this context is based on latent semantic analysis (LSA) that employs the statistical framework of singular value decomposition. Authors have also worked on graph-based techniques [5]. These techniques represent the documents with the help of analysis using graphical models. TextRank and LexRank are examples of these techniques. Further it was found out that extractive summarization is more robust as it uses the text span from source document. Here, the challenges encountered are due to their basic idea to select and join key words or phrases. In certain works, based on abstractive methods, it was seen that there is loss of information due to compression of text into fixed-length vectors [6] [7]. A Note summarization model has also been developed and presented by researchers. This helps combine notes of multiple users all over for students to study efficiently [8] [9]. Here, the entire framework comprises of four modules: one that finds supporting materials, topic identification, integrating learning materials and finally mapping of the content. Another generic abstract summarizer has also been presented in literature. This work first divides the given text

into various sections based on varied topics. Moreover, sentence reduction rule is also applied here.

From the above discussion, it becomes clear that many researchers, academicians, and students use multiple different type of documents for their daily academic activities. The text summarization techniques can also support transfer and summarization of important documents transferred with the help of Mobile Ad hoc Networks especially in emergency situations and military applications [10][11].

So, it becomes pertinent to provide a proper system that summarizes text in an efficient fashion. This will help the academicians etc. in fast compilation and study of documents.

## III. Methodology

This study use the Lamini Flan T5 language model to summarize papers using a Streamlit application. With 248 million parameters and fine-tuned on the T5 architecture, this sophisticated open-source model was developed by Google. An application for text summarization using Streamlit is built utilizing this language model. This program is helpful as it can read PDFs and provide brief descriptions of them.

### Setting Up the Environment

The necessary environment must be activated before constructing the Streamlit application. Installing the requisite libraries and dependencies, including Transformers, Torch, and Streamlit, is an essential component of the setup.

Also, the Lamini Flan T5 model is ten downloaded and stored locally. Advantage is that there is no need for any API keys for this open-source model. Moreover, librarys can also be downloaded to implement and run this code.

**Installing Packages required to Build Text Summarizer**

The first step is installing and importing few libraries which are as follows:

```
import streamlit as st
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.document_loaders import PyPDFLoader, DirectoryLoader
from langchain.chains.summarize import load_summarize_chain
from transformers import T5Tokenizer, T5ForConditionalGeneration
from transformers import pipeline
import torch
import base64
```

After this, some basic functions are imported.

```
import streamlit as st
```

In Python, importing Streamlit as st means that all functions used from the Streamlit library will be preceded by the name st.

Next, some libraries are imported

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.document_loaders import PyPDFLoader, DirectoryLoader
from langchain.chains.summarize import load_summarize_chain
```

For text summarization process, text splitters play an important role. They are useful for processing and analysing large documents. Also, they are pivotal in applications that involve text retrieval, summarization, or natural language processing. LangChain has several built-in text splitters that can: Split long documents into smaller chunks, transform documents to suit an application and finally combine, filter, and manipulate documents.

LanChain has certain requirements. Document loaders act like data connectors, fetching information from various sources, such as text files, web pages, or video transcripts, and converting it into a format that LangChain can understand. Document loaders of LangChain's document can navigate the web as a user would, opening a browser window in headless mode to interact with webpages. T5 is an encoder-decoder model and converts all NLP problems into a text-to-text format. It is trained using teacher forcing. This means that for training, we always need an input sequence and a corresponding target sequence. The input sequence is fed to the model using input_Ids.

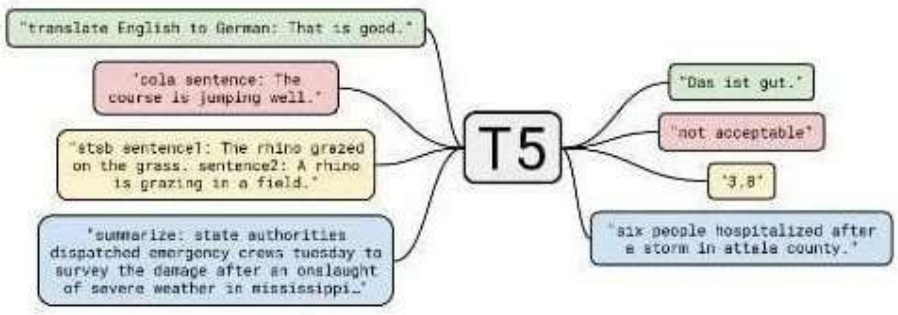



Fig. 1 Model Card for T5 base

In order to provide a straightforward application programming interface (API) that is dedicated to a variety of tasks, such as Named Entity Recognition, Masked Language Modelling, Sentiment Analysis, Feature Extraction, and Question Answering, these pipelines are objects that abstract the majority of the sophisticated code that is contained inside the library.

.

```
import torch
import base64
```

Torch-summary provides information complementary to what is provided by print (your_model) in PyTorch, similar to Tensorflow's model.

**Explanations**

```
#model and tokenizer loading
checkpoint = "LaMini-Flan-T5-248M"
tokenizer = T5Tokenizer.from_pretrained(checkpoint)
base_model = T5ForConditionalGeneration.from_pretrained(checkpoint,
                    device_map='auto', torch_dtype=torch.float32)
```

After all Libraries are import then we use this command to check Tokenizer loading. Basically, in this code its can checks the checkpoint via LaMini-Flan-T5-248M if they all are clear then its works on base model.

Next step:

**Composing and Posting the PDF Document**

```
#file loader and preprocessing
def file_preprocessing(file):
    loader =  PyPDFLoader(file)
    pages = loader.load_and_split()
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=200, chunk_overlap=50)
    texts = text_splitter.split_documents(pages)
    final_texts = ""
    for text in texts:
        print(text)
        final_texts = final_texts + text.page_content
    return final_texts
```

The program lets people upload PDF files to enable document summarizing. With Streamlit's file upload capabilities and the Langchain library's preprocessing capabilities, we can easily handle PDF files that are uploaded. The content of the PDF files will be extracted using the Langchain library's text splitting and document loading features.

**Next step:**

**Using LLM Pipeline for Summarize the Model.**

```
#LLM pipeline
def llm_pipeline(filepath):
    pipe_sum = pipeline(
        'summarization',
        model = base_model,
        tokenizer = tokenizer,
        max_length = 500,
        min_length = 50)
    input_text = file_preprocessing(filepath)
    result = pipe_sum(input_text)
    result = result[0]['summary_text']
    return result
```

**Next step:**

```python
#streamlit code
st.set_page_config(layout="wide")

def main():
    st.title("Document Summarization App using Langauge Model")

    uploaded_file = st.file_uploader("Upload your PDF file", type=['pdf'])

    if uploaded_file is not None:
        if st.button("Summarize"):
            col1, col2 = st.columns(2)
            filepath = "data/"+uploaded_file.name
            with open(filepath, "wb") as temp_file:
                temp_file.write(uploaded_file.read())
            with col1:
                st.info("Uploaded File")
                pdf_view = displayPDF(filepath)

            with col2:
                summary = llm_pipeline(filepath)
                st.info("Summarization Complete")
                st.success(summary)
```

In this code, the primary function is to ascertain the uploaded file's status. Upon file upload, the main function validates the file path. If the file upload is successful, the subsequent steps involve summarizing the content, displaying the summarized output, or initiating the pipeline for tasks such as filepath configuration, dataset preparation, or model fine-tuning. This comprehensive approach ensures the seamless execution of the summarization process, from file upload to generating concise summaries, thereby facilitating a robust and user-friendly experience.

## IV. Results

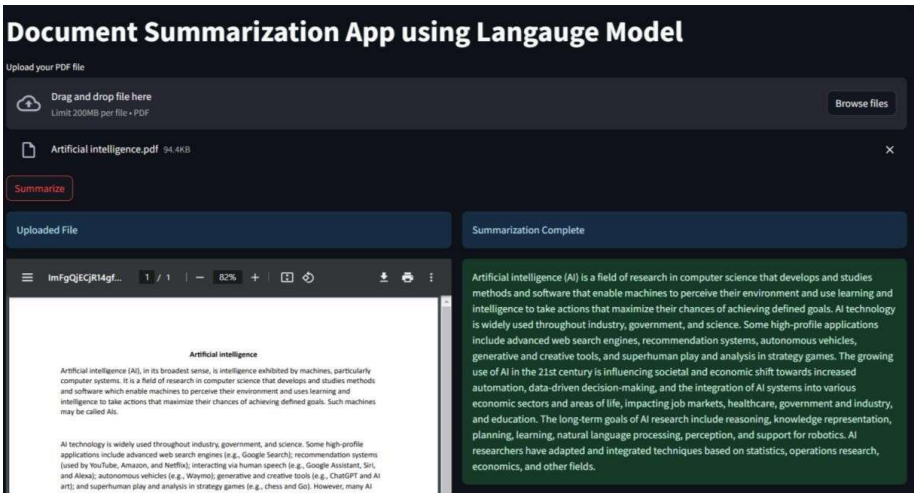
The result demonstrates a pdf document summarizer on your web browser.
Backend code result is as follows:

```
PS C:\Users\ayush\Desktop\Major\LAMINI_LLM_SUMMARISATION> streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://127.0.0.1:8501
```

Frontend code results:




## Conclusion:

The PDF Text Summarizer application, developed using the LaMini-Flan-T5-248M model and implemented via Streamlit, enables users to submit PDF documents and produce summaries. It leverages the power of natural language processing to extract key information from lengthy texts, making it a valuable tool for researchers, students, and professionals. By providing an efficient way to condense large amounts of content, the application enhances productivity and facilitates better understanding of complex documents. The users must remember to customize and fine-tune the summarization options based on their specific use case. To generate brief, accurate summaries of PDFs supplied, the app makes use of the model's robust capabilities. Research, content curation, and information retrieval are just a few areas where this technology might prove to be quite useful.

In future, the work can be extended to include multi document summarization as well as interactive user interface.

## References

[1] Watanangura, P., Vanichrudee, S., Minteer, O., Sringamdee, T., Thanngam, N., & Siriborvornratanakul, T. (2023). A comparative survey of text summarization techniques. *SN Computer Science*, *5*(1), 47.

[2] Cui, P., Hu, L., & Liu, Y. (2020). Enhancing extractive text summarization with topic-aware graph neural networks. *arXiv preprint arXiv:2010.06253*.

[3] Xie, F., Chen, J., & Chen, K. (2023). Extractive text-image summarization with relation-enhanced graph attention network. *Journal of Intelligent Information Systems*, *61*(2), 325-341.

[4] Sharma, G., & Sharma, D. (2022). Automatic text summarization methods: A comprehensive review. *SN Computer Science*, *4*(1), 33.

[5] Fang, C., Mu, D., Deng, Z., & Wu, Z. (2017). Word-sentence co-ranking for automatic extractive text summarization. *Expert Systems with Applications*, *72*, 189-195.

[6] Moradi, M., Dashti, M., & Samwald, M. (2020). Summarization of biomedical articles using domain-specific word embeddings and graph ranking. *Journal of Biomedical Informatics*, *107*, 103452.

[7] Prathap, G., & Rathinasabapathy, R. (2023). A text summarization hybrid approach using CNN and the firefly algorithm. *SN Computer Science*, *5*(1), 119.
[8] Kirmani, M., Kour, G., Mohd, M., Sheikh, N., Khan, D. A., Maqbool, Z., ... & Wani, A. H. (2024). Biomedical semantic text summarizer. *BMC bioinformatics*, *25*(1), 152.
[9] Deotale, R., Rawat, S., Vijayarajan, V., & Surya Prasath, V. B. (2021). POCASUM: policy categorizer and summarizer based on text mining and machine learning. *Soft computing*, *25*(14), 9365-9375.
[10] Rao, M., Chaudhary, P., Sheoran, K., & Dhand, G. (2023). A secure routing protocol using hybrid deep regression based trust evaluation and clustering for mobile ad-hoc network. *Peer-to-Peer Networking and Applications*, *16*(6), 2794-2810.
[11] Rao, M., & Singh, N. (2018). Energy efficient QoS aware hierarchical KF-MAC routing protocol in MANET. *Wireless Personal Communications*, *101*, 635-648.