A Report On

SummAI – Automated Business Text Summarization.

Submitted by

Pamulapati Sai Manvitha -AP23110010001

Cherukuri Venkata Arjun-AP23110010043

Maturi Pardha Saradhi-AP23110010056

Lingineni Prudhvi Sai-Ap23110010076

Koyyana Karthikeya-AP23110011269

CSE-U

BACHELOR OF TECHNOLOGY
IN

Computer Science and Engineering

School of Engineering and Sciences

Under the guidance of

Gen AI Faculty

Department of Computer Science

[December, 2025]

SRM University AP, Neerukonda, Mangalagiri, Guntur.

Andhra Pradesh - 522240

Abstract

This project, titled **SummAI – Automated Business Summarization**, focuses on building a hybrid text summarization system that can shorten long articles, reports and emails into meaningful short summaries. The main idea is to support busy users like business analysts, students and working professionals who need to quickly understand large amounts of text instead of reading everything line by line.

SummAI supports both extractive and abstractive summarization. The extractive method picks the most important sentences directly from the original text, while the abstractive method uses a transformer model (T5) to generate new sentences that still keep the original meaning. Users can also choose the summary length as short, medium or long, which gives some control over how detailed the summary should be.

For training and testing, I used the CNN/DailyMail dataset available on Kaggle, which is a well-known dataset containing news articles and their human-written summaries. I built a Flask-based web application that allows users to paste text or upload files in `.txt` or .pdf format. The summarizer then processes the input and shows the summary on the right side of the interface, with options to download it as a text file or a PDF.

Overall, this project helped me understand how natural language processing (NLP) works in a practical context. I learned the difference between extractive and abstractive summarization, how transformer models work at a high level, and how to deploy a simple machine learning based tool as a web application. The system is not perfect, but it works fairly well and can be improved further in the future.

Table Of Contents

1. Introduction

In the current digital world, we are surrounded by text everywhere. There are news articles, blog posts, research papers, business emails, project reports and many other forms of written content. Sometimes it genuinely feels like there is too much information and not enough time to read everything carefully. Because of this, text summarization has become a very important topic.

Summarization simply means taking a long piece of text and converting it into a shorter version that still keeps the key points. When done properly, a good summary saves time and reduces mental effort. Busy people like managers or analysts can quickly understand what is going on without reading every paragraph.

The main motivation behind SummAI is very simple. As a student, I often receive long PDFs, case studies and notes. Many times I honestly feel like skipping them because they are too long. I always thought it would be nice if some tool could just give me a few clear paragraphs that explain the main ideas. This personal feeling slowly turned into the idea for this project.

In this chapter, I will give a brief introduction to text summarization, types of summarization, and the basic goals of my project. I am not trying to be extremely formal here. Instead, I want to explain things in a way that feels natural and easy to follow, just like how I understood it while working on the project.

1.1 Background of Text Summarization

Text summarization has been studied for many years. Earlier, most methods were rule-based or purely statistical. They focused on word frequency, sentence position and some hand-written rules. Over time, machine learning based models came in, and now transformer models are very common.

There are mainly two types of summarization:

- **Extractive summarization** – selects the most important sentences from the original text and combines them to form a summary.
- **Abstractive summarization** – generates new phrases or sentences that may not appear in the original text but still carry the same meaning.

Extractive methods are usually easier to implement and faster. Abstractive models are more advanced and feel more like how humans summarize, but they need more computation and stronger models.

1.2 Motivation

Honestly, the motivation for this project came from real, everyday struggles. Whenever I see a 20-page report or a very long article, I feel a small sense of panic. It is not that I do not want to read. It is just that my brain gets tired, especially when I already have other assignments and tasks.

So instead of just complaining about long text, I decided to build something that could at least partially solve this problem. Also, generative AI has become very popular recently, and I wanted to do a project that actually uses it in a practical way.

1.3 Goals and Objectives

The main objectives of SummAI are:

- To build a web-based summarization tool that can handle long business-related text.
- To implement both extractive and abstractive summarization techniques.
- To allow users to choose the type of summary and the length (short, medium, long).
- To compare the outputs of extractive and abstractive methods in a simple and understandable way.
- To gain hands-on experience with NLP, transformer models, Flask and front-end development.

1.4 Scope of the Project

The current scope is focused on:

- English language text.
- Business-oriented content such as news articles, reports and emails.
- A single-user web application, mainly for demo and learning purposes.

The system is not meant to be a large-scale production system. It is more like a working prototype that shows what is possible with relatively simple code and publicly available models.

2. Project Description

2.1 Overview of SummAI

SummAI is a hybrid text summarization system that tries to combine the strengths of both extractive and abstractive summarization. Instead of choosing only one type, the idea is to give the user the flexibility to decide what they need in that moment.

The user interacts with the system through a web interface. On the left side, there is an input area where they can either paste text or upload a file. On the right side, the summarized output is displayed. At the top, there is a simple mode selector where the user can switch between extractive and abstractive modes. There is also a drop-down for selecting the summary length.

2.2 Problem Statement

The problem statement can be described as follows:

> Build a summarizer that can shorten long business articles, reports or emails into meaningful short summaries. Compare extractive vs. abstractive methods.

This statement might look very short, but it captures the essence of the problem quite clearly. Long documents are hard to consume, and people want shorter versions. At the same time, there is more than one way to summarize, and each method has pros and cons. So comparison is also part of the project.

2.3 Key Features of the System

Some of the main features of SummAI are:

- Support for both **extractive** and **abstractive** summarization.
- **Three summary lengths**: short, medium and long.
- Option to either **paste custom text** or **upload a text/PDF file**.
- A clean, minimal **dark/light mode UI**.
- Ability to **download the summary** as a text file or a PDF file.
- Clear highlighting of sentences in extractive mode to show which parts were selected.

2.4 Target Users

The tool is mainly aimed at:

- Business analysts and managers.
- Students reading case studies and research papers.
- Anyone who wants a quick summary of a long article.

Of course, it is just a prototype, but even in this early stage it can already be useful in simple scenarios.

2.5 Personal Perspective

From a personal point of view, this project is a nice mix of theory and practice. On one side, it involves NLP and transformer models, which are quite trendy and powerful. On the other side, it includes basic web development and user experience design. Working on it gave me a feeling of building a complete mini product rather than just writing some code snippets.

3. Project Scenarios

In this chapter, I describe some realistic situations where SummAI can be useful. These scenarios helped me design the project in a more practical way, instead of thinking only in terms of theory.

3.1 Scenario 1: Business Analyst Tracking Market News

Imagine Janani, a business analyst, who has to track industry news and competitor activities every day. She opens multiple news websites, each having several long articles. Reading every article fully takes a lot of time, and sometimes she only needs the high-level idea.

Using SummAI, she can just copy-paste the article content into the input box or upload the article if it is saved as a file. She selects the summarization mode and gets a nice summary within a few seconds. This helps her quickly scan through different articles and decide which ones deserve more detailed attention.

3.2 Scenario 2: Summarizing Long Emails

In many organizations, email threads become very long. There might be a lot of back-and-forth communication, and someone joining the thread later may struggle to understand the full context.

In this scenario, a user can copy the email content and use SummAI to generate a short summary. They might choose a short-length extractive summary first, just to get the key decisions and outcomes. If needed, they can then switch to a longer, abstractive summary for more details.

3.3 Scenario 3: Student Revising Research Papers

Students often struggle while revising research papers before exams or presentations. Many times, they only need to remember the main contributions, methods and results rather than every single detail.

Here, SummAI can help by producing a clear and readable summary of the paper. The student can then go back to the original document to re-read only the important sections.

3.4 Scenario 4: Quick Meeting Preparation

Sometimes a manager has to prepare for a meeting on short notice. They might receive a long report or presentation deck just one hour before the meeting. Instead of reading every slide or paragraph, they can run the document through SummAI and get a short list of key points.

This is not a perfect replacement for fully reading the report, but it at least gives them something to start with and reduces stress.

4. Prerequisites

4.1 Software and Tools

To implement SummAI, the following tools and software were used:

- **Python 3.8+** – core programming language.
- **Flask** – web framework for creating the backend.
- **Transformers** (Hugging Face) – for loading the T5 model.
- **NumPy** and **scikit-learn** – for numerical operations and cosine similarity.
- **PyPDF2** – to extract text from PDF files.
- **HTML, CSS, JavaScript** – for the frontend design and interactivity.
- **Jupyter Notebook or VS Code** – for experimentation and development.

4.2 Python Libraries

Some important Python libraries used are:

- transformers
- numpy
- sklearn
- re for regular expressions
- flask
- PyPDF2

4.3 Dataset Details

The project uses the CNN/DailyMail news summarization dataset from Kaggle. It includes a large number of news articles and their corresponding summaries.

https://www.kaggle.com/datasets/gowrishankarp/newspaper-text-summarization-cnn-

In practice, for running and testing locally, I did not use the entire dataset at once because it is fairly large. Instead, I experimented with a smaller subset of articles to understand how well the summarizer performs.

4.4 Knowledge Prerequisites

Before starting this project, it is helpful to have a basic understanding of:

- Python programming.
- Basic machine learning terms.
- What NLP is and why it matters.
- Some idea about neural networks and transformers (even at a high level).

I did not know everything perfectly when I started. I learned many things on the way, especially about T5 and how summarization models are fine-tuned.

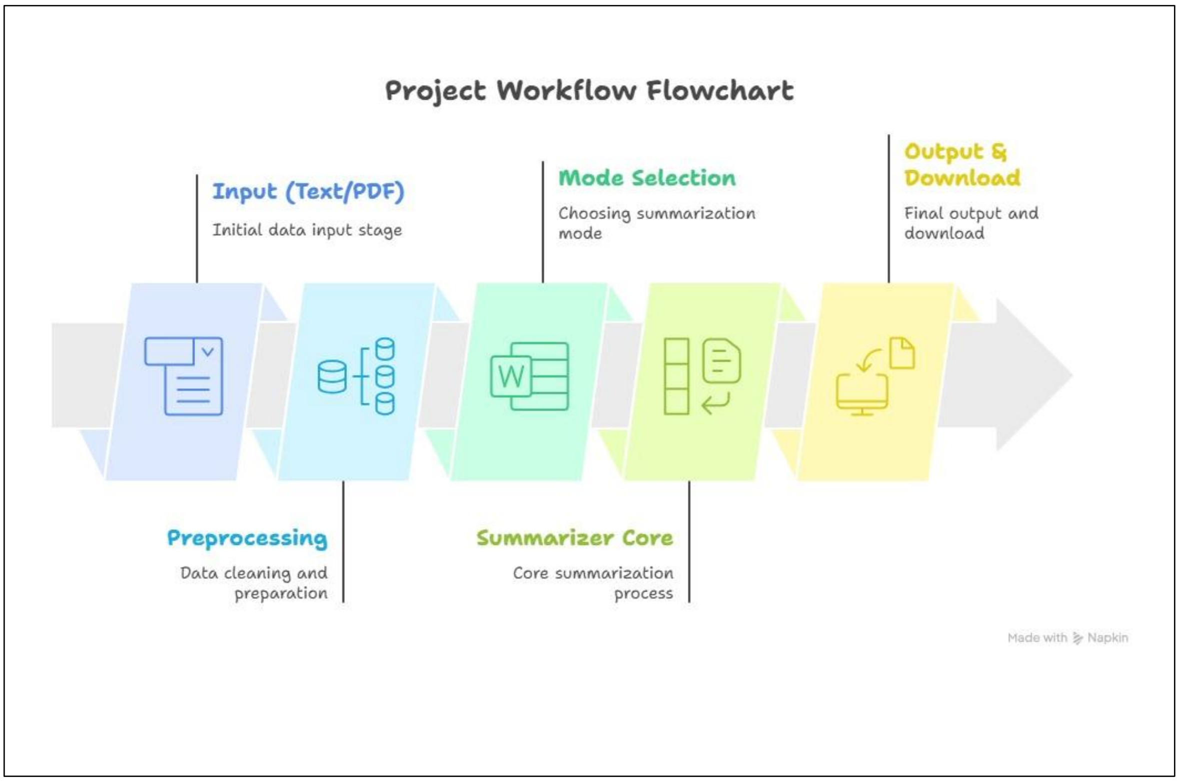
5. Project Flow

5.1 Overall Workflow

The overall workflow of SummAI can be summarized in a few steps:

1. User opens the SummAI web page.
2. User either pastes custom text or uploads a file.
3. User selects the summarization mode (extractive or abstractive).
4. User selects summary length (short, medium or long).
5. Flask backend receives the text and passes it to the summarizer.
6. Summarizer generates the summary.
7. Summary is displayed on the right with options to download.

5.2 Flow Diagram Placeholder


Project Workflow Flowchart
Input (Text/PDF)
Initial data input stage
Mode Selection
Choosing summarization mode
Output & Download
Final output and download
Preprocessing
Data cleaning and preparation
Summarizer Core
Core summarization process
Made with Napkin

5.3 Step-by-Step Description

5.3.1 Input Handling

The system supports two main types of input:

- Raw text pasted into the text area.
- File upload (.txt or .pdf).

If a file is uploaded, the backend reads the text from it. For PDF files, PyPDF2 is used to extract text page by page.

5.3.2 Preprocessing

The input text goes through some basic preprocessing:

- Removing extra newlines and spaces.
- Normalizing dashes and special characters.
- Splitting into sentences.

The preprocessing is not extremely advanced, but it is enough for the current version.

5.3.3 Summarization Core

The summarization core has two main branches:

- Extractive summarization using a TextRank-like approach.
- Abstractive summarization using the T5 model.

Based on the mode selected by the user, the appropriate function is called.

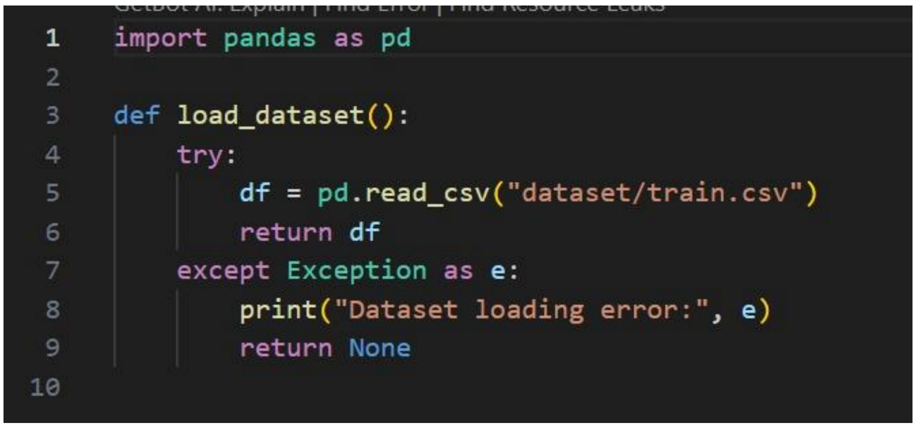
5.3.4 Output Presentation

The summary is split into sentences and displayed neatly. In extractive mode, each important sentence can be slightly highlighted to show that it was chosen by the algorithm. The user can then decide to download the summary if needed.

6. Milestone 1: Data Collection and Preparation

6.1 Loading the Dataset

A helper script called dataset_loader.py is used to load the dataset. It simply reads a CSV file from the dataset folder and returns a pandas DataFrame.



```python
import pandas as pd

def load_dataset():
    try:
        df = pd.read_csv("dataset/train.csv")
        return df
    except Exception as e:
        print("Dataset loading error:", e)
        return None

```

6.2 Understanding the Data Structure

The dataset mainly has:

- article – the full news article.
- highlights – the human-written summary.

These pairs are useful for checking how close our generated summaries are to the real human summaries.

6.3 Basic Cleaning

Some of the cleaning steps include:

- Removing very short or empty articles.
- Replacing multiple spaces with a single space.
- Removing some unwanted characters or markers if present.

Even though the dataset is already well-structured, small cleaning steps help improve summarization quality.

6.4 Challenges Faced

One small challenge was the size of the dataset. It is large enough that loading the full CSV and processing all rows at once is not convenient on a normal laptop. So I ended up sampling a portion of the data for experiments. This is okay for a project-level implementation.

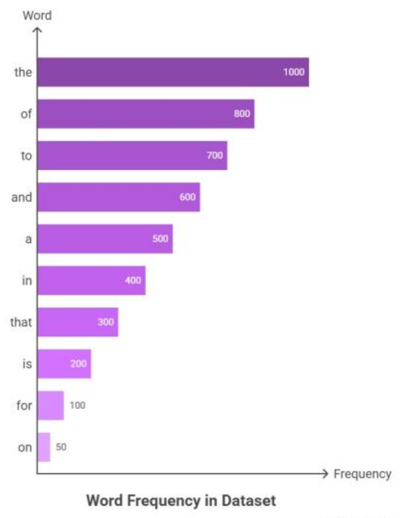
7. Milestone 2: Exploratory Data Analysis

7.1 Text Length Distribution

As part of EDA, I checked how long the articles are. Some articles are just a few paragraphs, while others are quite long. This matters because very short text does not need heavy summarization, while very long articles definitely benefit from it.

7.2 Common Words and Phrases

If we generate a word frequency plot, common words like "said", "news", "people" and so on appear often. However, many of these are stopwords or general terms. For summarization, what matters more are the informative words and entities.



7.3 Observations

From basic EDA, a few observations can be made:

- Many articles follow a typical news writing pattern.
- The human-written summaries are usually short and to the point.
- There is sometimes a gap between article length and summary length, which confirms that summarization is really needed.

This exploration step gave me confidence that building a summarizer on this dataset makes sense.

8. Milestone 3: Extractive Model Design

8.1 Sentence Tokenization

In the extractive model, the first step is to break the text into sentences. I used regular expressions to split by punctuation like full stops, question marks and exclamation marks. Some basic cleaning is also done to remove empty sentences.

8.2 Vector Representation of Sentences

To measure similarity between sentences, each sentence is converted into a bag-of-words vector. The vocabulary is built from all words in the text. Each sentence gets a vector where each dimension represents a word count.

8.3 Similarity Matrix and TextRank

Once the sentence vectors are ready, a similarity matrix is computed using cosine similarity. Then a simple PageRank-like algorithm is run on this matrix. Sentences that are similar to many other sentences get higher scores.

8.4 Sentence Selection

Based on the number of sentences and the chosen length (short, medium or long), a certain percentage of top-ranked sentences is selected as the summary. The sentences are then sorted in their original order and joined together.

8.5 Advantages and Limitations

Advantages:

- Easy to implement.
- Fast and does not require heavy computation.

Limitations:

- Sometimes the summary feels a bit choppy.
- It may repeat similar sentences.
- It does not rewrite or simplify complex sentences.

9. Milestone 4: Abstractive Model Design

9.1 Using T5 for Summarization

For the abstractive part, I used the t5-small model from the Hugging Face library. T5 is a sequence-to-sequence transformer model that can perform many NLP tasks just by changing the input prompt.

In my project, I constructed prompts like:

- "summarize in one short sentence:"
- "summarize in a detailed paragraph:"

These prompts help guide the model to generate summaries of different lengths.

9.2 Handling Length Variations

Depending on whether the user selects short, medium or long, different settings for maximum and minimum lengths are passed to the T5 generation function. For example:

- Short: 8 to 40 tokens.
- Medium: 20 to 80 tokens.
- Long: 50 to 150 tokens.

These are not perfect but give reasonable control over the output size.

9.3 Beam Search and Repetition Control

Beam search with multiple beams is used to explore more possible summaries than a simple greedy approach. Also, a no_repeat_ngram_size option helps avoid repeated phrases in the summary.

9.4 Strengths and Weaknesses

Strengths:

- Generates more natural and human-like summaries.
- Can compress content significantly while keeping the main idea.

Weaknesses:

- Slower than extractive models.
- Sometimes misses small factual details.
- Needs a good amount of memory and model loading time.

10. Milestone 5: Web Application and UI

10.1 Flask Backend

The main backend file is app.py. It defines the routes and handles requests.

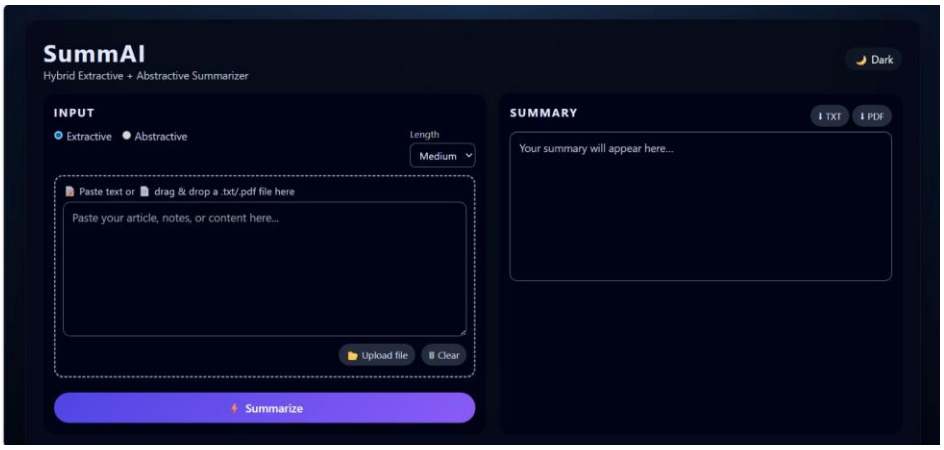
- The root route (/) supports both GET and POST.
- On POST, it checks whether the user has submitted custom text or uploaded a file.
- It reads the text, calls summarize_text() from summarizer.py and passes the result to the template.

10.2 Frontend Structure

The HTML file defines two main sections side by side:

- **Input section** on the left.
- **Output section** on the right.

There is also a header with the project name and a button to toggle between dark and light mode.


SummAI
Hybrid Extractive + Abstractive Summarizer
Dark
INPUT
Extractive
Abstractive
Length
Medium
Paste text or drag & drop a .txt/.pdf file here
Paste your article, notes, or content here...
Upload file
Clear
Summarize
SUMMARY
TXT
PDF
Your summary will appear here...

10.3 JavaScript Interactions

The script.js file handles:

- Theme toggle (dark/light).
- Drag-and-drop area for file uploads.
- Clearing the input text.
- Downloading the summary as TXT or PDF.

10.4 CSS Styling

The style.css file is responsible for the modern look of the interface. It defines:

- Layout grid.
- Colors for dark and light themes.
- Button designs.
- Highlight styles for summary sentences.

Even though it is not a very advanced design, it makes the app look clean and pleasant compared to a plain HTML page.

11. Results and Discussion

11.1 Example Outputs

In this chapter, I show some example results and discuss them informally.

11.1.1 Example 1: News Article

Input: A long news article describing a major corporate merger, around 800–1000 words.

Extractive Summary: The extractive version picked around 4–5 sentences that mainly covered the names of the companies involved, the financial value of the deal and the expected benefits.

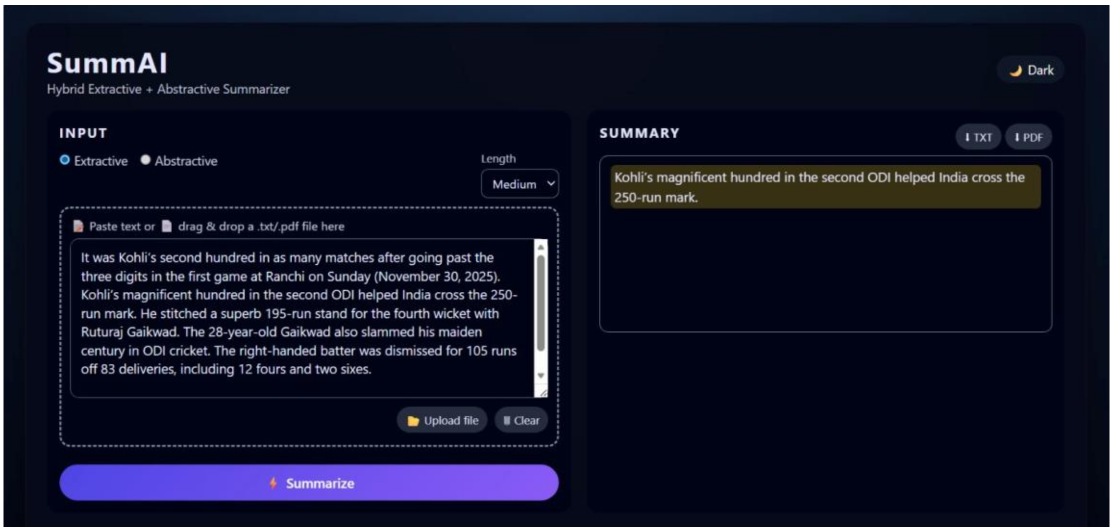
Abstractive Summary: The abstractive summary rewrote the content into 2–3 smooth sentences that mentioned the merger, the reason behind it and its impact. It felt more like a human-written summary.

11.1.2 Example 2: Long Email

Input: A long email thread with multiple replies.

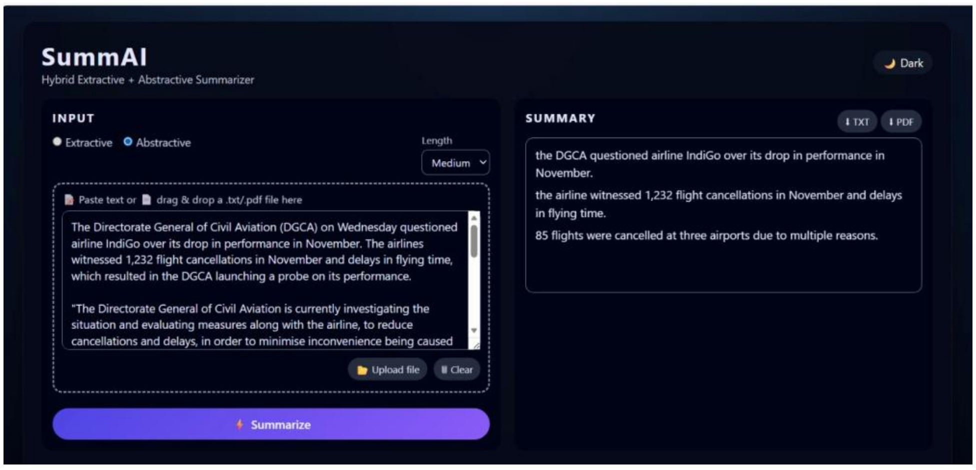
Extractive Summary: It picked sentences that included key decisions and dates. However, it also included a bit of redundant information.

Abstractive Summary: This one tried to summarise the overall discussion into a single paragraph. Sometimes it slightly changed the wording, but the meaning stayed correct.


SummAI
Hybrid Extractive + Abstractive Summarizer
Dark
INPUT
Extractive
Abstractive
Length
Medium
Paste text or drag & drop a .txt/.pdf file here
Upload file
Clear
Summarize
SUMMARY
TXT
PDF

It was Kohli's second hundred in as many matches after going past the three digits in the first game at Ranchi on Sunday (November 30, 2025). Kohli's magnificent hundred in the second ODI helped India cross the 250-run mark. He stitched a superb 195-run stand for the fourth wicket with Ruturaj Gaikwad. The 28-year-old Gaikwad also slammed his maiden century in ODI cricket. The right-handed batter was dismissed for 105 runs off 83 deliveries, including 12 fours and two sixes.

Kohli's magnificent hundred in the second ODI helped India cross the 250-run mark.



Length: Medium

Paste text or drag & drop a .txt/.pdf file here

The Directorate General of Civil Aviation (DGCA) on Wednesday questioned airline IndiGo over its drop in performance in November. The airlines witnessed 1,232 flight cancellations in November and delays in flying time, which resulted in the DGCA launching a probe on its performance.

"The Directorate General of Civil Aviation is currently investigating the situation and evaluating measures along with the airline, to reduce cancellations and delays, in order to minimise inconvenience being caused

the DGCA questioned airline IndiGo over its drop in performance in November.

the airline witnessed 1,232 flight cancellations in November and delays in flying time.

85 flights were cancelled at three airports due to multiple reasons.

11.2 User Experience Feedback (Informal)

I asked a couple of friends to use the tool quickly. Their feedback was:

- The interface is simple and not confusing.
- The dark mode option is nice to have.
- Summaries are helpful, especially for long text.
- Abstractive summaries feel smoother.

Of course, this is not a formal user study, but it gave me some confidence that the project is understandable and usable.

12. Comparison of Extractive and Abstractive Methods

12.1 Qualitative Comparison

Based on my observation, extractive summaries are more accurate in terms of not changing facts, because they literally copy sentences from the original text. But they sometimes sound like a list of disconnected lines.

Abstractive summaries, on the other hand, are more natural and flow better. They feel closer to how a human would write a summary. However, in rare cases they might miss small details or rephrase something in a slightly different way.

12.2 Simple Comparison Table

Criterion	**Extractive**	**Abstractive**
Speed	Faster	Slower
Grammar	Same as original	Usually smooth and natural
Risk of changing meaning	Very low	Slightly higher
Readability	Medium	High
Implementation effort	Lower	Higher

12.3 Personal View

Personally, I like the abstractive mode more because it gives a nice feeling when the model rewrites everything in a cleaner way. But I also appreciate the extractive mode, especially when I want to be sure that no detail is rephrased incorrectly. So I think having both modes in one tool is actually a good idea.

13. Limitations and Future Enhancements

13.1 Current Limitations

Some of the main limitations of the current version of SummAI are:

- It is limited to English text.
- It is tested mainly on news articles and some informal content.
- The abstractive summarizer may be slow on very large documents.
- The system does not yet provide confidence scores or any automatic quality measure for the summary.
- There is no support for saving user history or summaries.

13.2 Possible Improvements

In the future, the project can be improved by:

- Adding support for other languages.
- Allowing batch summarization of multiple documents.
- Integrating more advanced models or fine-tuning on specific business datasets.
- Adding evaluation metrics like ROUGE directly in the interface.
- Providing an option to highlight which parts of the original text contributed most to the summary.

13.3 Deployment Possibilities

Right now, the project runs locally through Flask. In the future, it could be:

- Deployed on a cloud platform.
- Integrated into a browser extension for instant webpage summarization.
- Extended into a mobile app for on-the-go reading.

These ideas are slightly ambitious, but they show that the project has room to grow.

14. Conclusion and Personal Reflection

14.1 Conclusion

SummAI started as a simple idea to make long texts less scary and more manageable. Over time, while building this project, I got a chance to combine several different skills: Python coding, working with datasets, using transformer models, and designing a small web application.

The project successfully demonstrates:

- How extractive summarization can be implemented using a TextRank-like approach.
- How abstractive summarization can be done using a pre-trained T5 model.
- How to combine these techniques into a user-friendly web tool.

While the system is not perfect, it works reasonably well for news articles and long text inputs. It clearly shows the difference in style and behavior between extractive and abstractive summarization.

14.2 What I Learned

On a more personal note, I learned:

- That NLP is not as scary as it looks at first, if you break it down step by step.
- How transformers and pre-trained models can be used without fully understanding every mathematical detail.
- That front-end work (HTML, CSS and JS) also matters a lot, because it directly affects how others experience the project.
- How to debug small issues like path problems, model loading errors and PDF reading errors.

There were moments when the code was not working, and I honestly felt like giving up. But slowly fixing each issue and finally seeing everything work together was a very satisfying feeling.

14.3 Final Thoughts

Overall, I am happy with what I managed to build with SummAI. It might not be a production-level system, but as a student project it ticks many boxes: it is practical, it uses modern AI techniques, it has a working UI and it solves a real problem that people actually face with long texts.

References

- CNN/DailyMail News Summarization Dataset, Kaggle. https://www.kaggle.com/datasets/gowrishankarp/newspaper-text-summarization-cnn-dailymail
- Hugging Face Transformers Documentation. https://huggingface.co/docs/transformers
- Flask Documentation. https://flask.palletsprojects.com
- T5 Model Paper – “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”.
- Various online tutorials and community answers on StackOverflow for debugging and implementation help.