PDF Summarization Tool Using NLP and AI

Muhammad Sarim Usmani

Date: September 24, 2024

Gisma University of Applied Sciences

Contents

List of Figures

Chapter 1

Problem Statement

1.1 Business Problem

The customer organization, which is a news aggregator, is in the need to process and summaries large quantities of textual information extracted from PDFs. The main issue is that customers require a quick search of these publications' content with no necessity to read through the entire material. This means that for the user to be quickly able to understand what the key concepts are, an application that is based on NLP and AI must have the ability to come up with summaries by themselves.

1.1.1 Importance of the Problem

Today people have a way of accessing more information and what they need is how to search for the particular information and come up with the right result. This tendency can be described as 'information overload' becoming a severe problem for both people and enterprises. Consumers are overwhelmed with a vast number of texts that come from different sources such as news articles, research papers, reports, etc. There is an incredibly large load on people who try to make sense of all of this information, which results in lower efficiency and decision-making, otherwise known as decision fatigue.

Organizations must be able to quickly and clearly communicate important information in order to maintain users' satisfaction and engagement. People's attention spans are getting shorter, therefore there's a good probability more competent competitors will take their position. Thus, an AI-driven summarization tool is not just a convenience but also a need to guarantee that consumers can effectively navigate this sea of information. The tool provides summaries that highlight the key points so users may rapidly grasp the major concepts of papers without having to read through unnecessary details. In addition to saving time, users can focus on higher-level tasks like strategic planning, problem-solving, and decision-making.

Figure 1.1: Proposed Summarizer

1.2 Benefits of the Solution

The proposed NLP-based AI application offers extremely advantageous abstractive and extractive summaries that may be tailored to meet a variety of user needs. This dual approach boosts the value of the platform by offering customers the choice between an original, concise summary (abstractive) or an extended, verbatim excerpt (extractive). Operational efficiencies are enhanced by automating the summarizing process, since it significantly decreases the amount of manual work needed for content processing. As a result, the customer's company can handle larger data quantities and save costs without compromising quality. Scalability allows the platform to remain competitive while accommodating increasing user needs, which in turn increases user satisfaction and retention.

1.3 Data Collection

The effectiveness of this AI application is naturally determined by the quantity and type of textual data it learns from. The main data sources are the text crawled from user-submitted PDFs and news over web. These will be the sources that are used by both abstractive and extractive summarization techniques where this material must obtain. With document formats and content structures so diverse, extracting text thoughtfully is a must. The application needs to know how to handle multiple sorts of PDFs (scans, different webpage layouts), and be able extract the text in order for it to summarize correctly. Having a potential to collect data in this manner opens the user input options and allows application to perform consistently for wide range of inputs which ensures its reliability and build users trust.

Chapter 2

System Design

2.1 System Overview

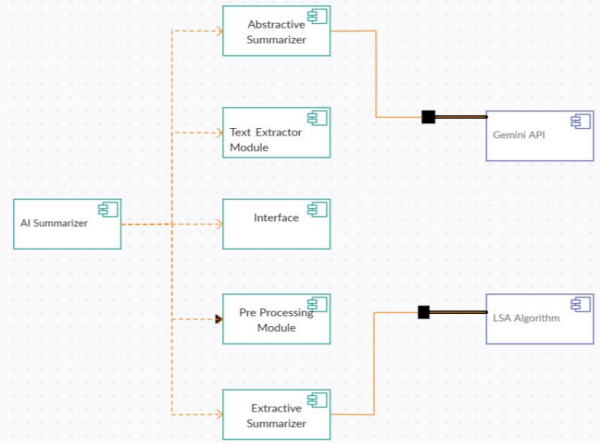

Figure 2.1: System Architecture Overview

The AI-driven PDF Summarization Tool is designed with several key components, each fulfilling a specific role in the system:

1. **User Interface:** It is a web-based tool with an easy-to-use front-end where users can upload PDF files or input URLs to online articles. The selection of a summarization's method, abstractive vs. extractive has been made quite intuitive for the user through this interface. The interface also gives the user real-time feedback of where each processing in summarizer stage helped keep users engaged and satisfied.

2. **Text Extraction Module:** Text Extraction Module – The heart of the tool, this module allows easy and precise text extraction across multiple sources. This module extracts text properly even from complex pdf formats, scanned documents and various webpage layouts. With its sturdy build, it can adapt to various type of documents, so you get the same reliable performance no matter what your input source is.

3. **Pre-processing Module:** After the text is scraped, we have Pre-processing Module which will perform necessary operations such as cleaning of data or tokenization and normalization. This module cleans the raw text from any irrelevant data, corrects some formatting messy issue and structures it to be an optimal input format for summarization. Pre-processing process is one of the very important steps which directly matters in building up a good summary with best quality and accuracy.

4. **Summarization Module:** The Summarization Module is the true core of the application. It puts at users' disposal the power of two condensed, very powerful summarization techniques: abstractive and extractive. Depending on the choice of the user, it will either generate a terse, original summary by rephrasing and synthesizing information or just select key sentences directly from the text. This flexibility makes this tool useful and attractive to the target audience by fitting different user needs and preferences.

5. **Backend Infrastructure:** At the very bottom lies a strong Backend Infrastructure that holds together the whole system and takes care of the smooth interaction among its constituents—the front-end interface and different modules for processing. It oversees an efficient data flow, processes user requests, and coordinates summary generation so that summaries are presented before the user in time and accurately. Besides, it is designed to scale so that rising volumes of data and corresponding user requests are handled seamlessly—without loss of performance.

2.2 Component Connection

- **User Interface to Backend:** The User Interface sends the uploaded PDF file or article URL as well as the user's chosen summarization method to the backend, which this connection initiates.

- **Backend to Text Extraction Module:** The backend receives the PDF or URL and passes it to a Text Extraction Module. This module further processes the document to extract relevant texts that need to be extracted for further stages of summarization.

- **Text Extraction to Preprocessing:** The text extracted is then fed into the Preprocessing Module, where the text should be cleaned and prepared, so it is ready to summarize.
- **Preprocessing to Summarization:** The text, after being pre-processed, is then fed into the Summarization Module. Depending upon the choice made by the user, the summarization process either involves the abstractive methods that create new summaries or the extractive methods that simply pick sentences.
- **Summarization to User Interface:** The final summary, generated by the Summarization Module, is then sent back to the User Interface to show the user, hence completing the summarization process and consequently open for user review.

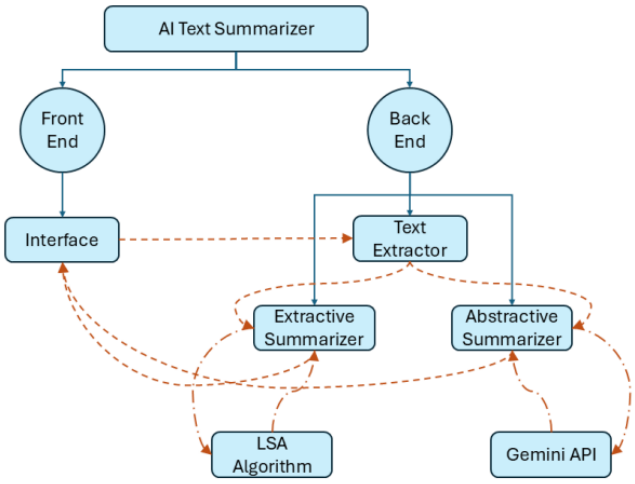

Figure 2.2: Connection between Components

2.3 Necessity of Each Component

- **User Interface:** The user interface is crucial for making it easy and straightforward for users to operate the application. Through this, users can upload a PDF or enter a URL for review, select their preferred summarization method, and then get the final summary. Otherwise, without it, an application will have no accessibility and usability features.
- **Text Extraction Module:** This module is important in that it transforms the content of PDF files into a form of text best suited for summarization. In the case

of availability of a URL, this module will parse and extract text from an online source, ensuring that all the content necessary to be processed is accessible.'

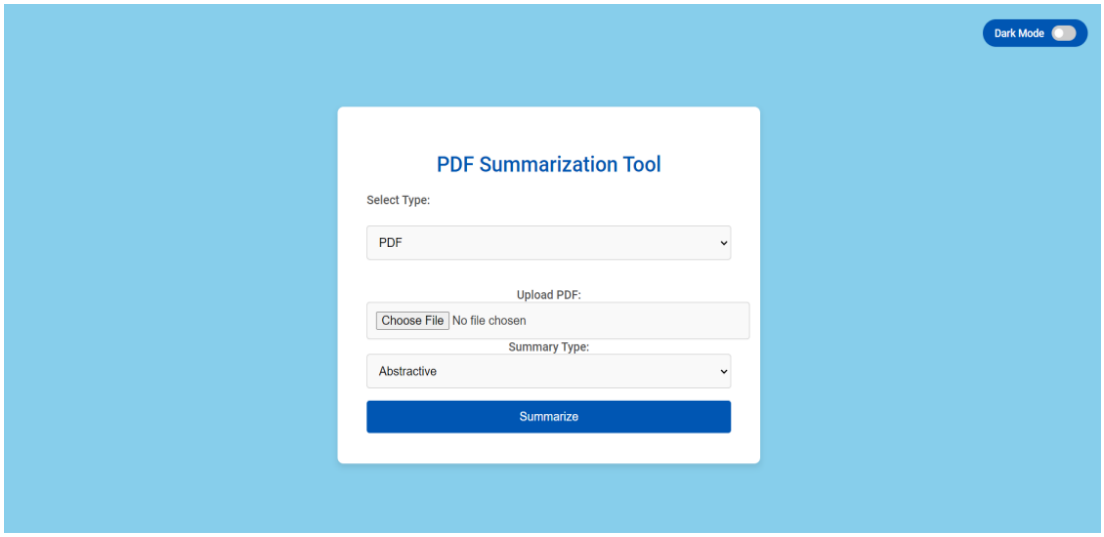

Figure 2.3: Interface of the Web App

- **Preprocessing Module:** The Preprocessing Module is important for increasing the quality of summaries. This module cleans the extracted text by removing noise and getting rid of irrelevant information, ensuring that only relevant information reaches the summarization stage for creating more accurate and useful summaries.
- **Summarization Module:** At the core of the system in generating summaries that directly relieve the business problem is the Summarization Module. This component, being at the core, is an extremely important one because it transforms pre-processed text into a meaningful but less wordy summary, therefore offering valuable insights from large volumes of information to users.

Chapter 3

Detailed Design, Implementation and Analysis

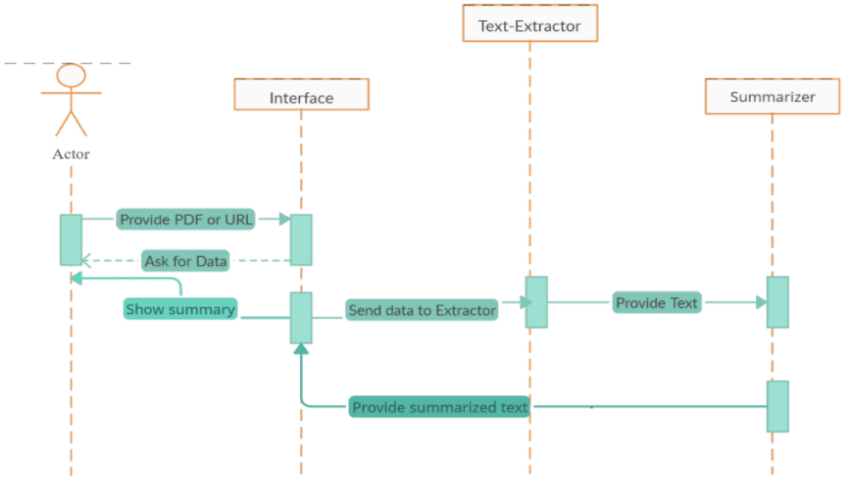

Figure 3.1: Detailed Design and Working

3.1 Text Extraction Module

Implementation:

- The extract-text-from-pdf function from PyPDF2 is used here for extracting text from PDFs. It goes through the PDF, parsing each page, aggregating the text into one unified string.

```python
from PyPDF2 import PdfReader

def extract_text_from_pdf(pdf_path):
    reader = PdfReader(pdf_path)
    text = ""
    for page in reader.pages:
        text += page.extract_text()
    return text
```

Figure 3.2: Function "extract-text-from-pdf"

- A function named fetch-article-text of newspaper3k is involved, used mainly to extract text from articles of a number of websites. The function is oriented towards webpage contents and ensures the grabbing of a relevant set of text to be processed.

```python
from newspaper import Article

def fetch_article_text(url):
    article = Article(url)
    article.download()
    article.parse()
    return article.text
```

Figure 3.3: Function "fetch-article-text "

Analysis:

- This method is effective for extracting text from a broad range of PDF documents and online articles, providing a robust solution for most standard formats.
- The approach may encounter difficulties with complex document layouts or non-standard fonts, which can affect the accuracy and completeness of the extracted text.

3.2 Preprocessing Module

Implementation:

- The Preprocessing Module performs some of the most vital tasks in text preparation. It cleans the text by removing special characters, HTML tags, extra whitespaces. This module also gets rid of stop words; all other words are lemmatized using NLTK's WordNetLemmatizer to standardize them.

```python
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import WordNetLemmatizer

#import nltk
#nltk.download('punkt')
#nltk.download('stopwords')
#nltk.download('wordnet')

def clean_text(text):
    # Remove HTML tags
    text = re.sub(r'<[^>]+>', '', text)
    # Remove special characters and digits
    text = re.sub(r'[^a-zA-Z\s]', '', text, re.I|re.A)
    # Convert to lowercase
    text = text.lower()
    # Remove extra whitespaces
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def preprocess_text(text):
    print('Started pre-processing')
    text = clean_text(text)
    print('Text cleaned')
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()

    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word.lower() not in stop_words]
    final =  " ".join(tokens)
    return final
```

Figure 3.4: Pre-Processing Module

Analysis:

- This module provides better quality input to summarization algorithms by focusing on relevant content. Additional use of NLTK will make this text processing robust and language-aware, increasing the accuracy and effectiveness of the summarization results.

3.3 Summarization Module

Implementation:

- **Abstractive Summarization:** This uses Google's Gemini API, and the process involves the creation of summaries like a human because it understands the context of the text; it creates content originally. It is flexible, readable, and completely dependent on the quality and settings used for the AI model.

```python
import google.generativeai as genai
import os
from dotenv import load_dotenv

load_dotenv()
key = os.getenv('GEMINI_API_KEY')
genai.configure(api_key = key)
generation_config = {'temperature' : 0.9, 'top_p' : 1, 'to_k' : 1, 'max_output_tokens': 2048}
model = genai.GenerativeModel('gemini-1.5-flash')


def abstractive_summary(text):
  response = model.generate_content([f'Write summary of the following text without bolding text: {text}'])
  return response.text
```

Figure 3.5: Abstractive Summarization Module

- **Extractive Summarization:** It uses sumy's LSA algorithm for extracting important sentences based on latent semantic analysis of text. It is a pretty basic way and surer with respect to the factual content as it has weak performance in contextualizing subtle pieces of information or rephrasing them properly.

```python
import sumy
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lsa import LsaSummarizer

def extractive_summary(text, num_sentences=5):
    parser = PlaintextParser.from_string(text, Tokenizer("english"))
    summarizer = LsaSummarizer()
    summary = summarizer(parser.document, num_sentences)
    return ' '.join([str(sentence) for sentence in summary])
```

Figure 3.6: Extractive Summarization Module

Analysis:

- Abstractive Summarization is capable of generating flexible and human-like summaries; it can thus particularize into the coherent, contextually rich content generation. However, it still relies on the robustness of the underlying AI model.
- Extractive Summarization is reliable and clear, picking up important sentences straight away from the text. It works fine for easy, straightforward factual content but misses nuance or the capacity for rewording.

3.4 Evaluation

3.4.1 System Evaluation:

Abstractive Summarization: Abstractive summarization will be graded based on coherence, readability, and the extent to which the core message is conveyed. The model represents the main ideas expressed in the text very well, is good at generating coherent and readable summaries. It sometimes generates summaries that lack detail or misinterprets complex content if nuanced information is critical.

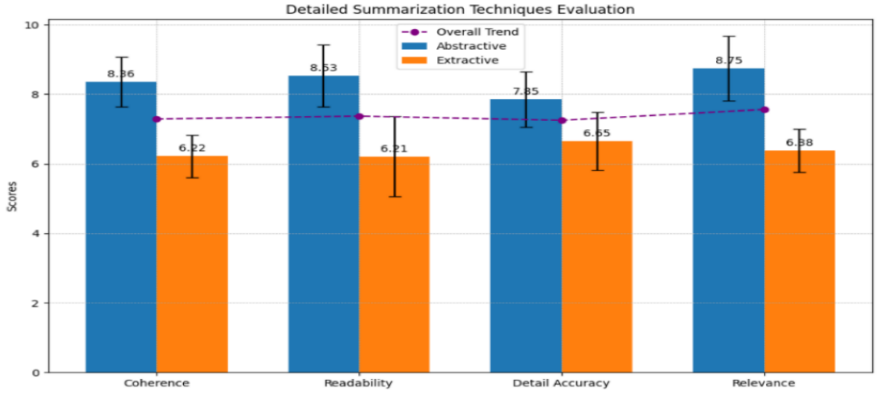

Figure 3.7: System Evaluation

Extractive Summarization: Extractive summarization is evaluated on the basis of accuracy of key sentences extracted and how relevant they are in describing the whole document. It functions well in texts that are relatively straightforward and whose main point is expressed explicitly. However, it may be poorly applicable to content where the critical information is implicit rather than stated and might therefore often miss such subtler aspects when summaries have to be prepared.

3.4.2 Component Evaluation:

Text Extraction: Text Extraction Module—a module that shows very high accuracy in rendering text from PDF documents and online articles with minimal loss or distortion. However, it does have some scope for further improvement regarding the handling of complex document structures, like multi-column or mixed content types. In this regard, an improved handling of such intricate formats by the module could further boost its reliability and robustness.

Preprocessing: It cleans and lemmatizes the extracted text to ensure that only quality input is fed into summarization models. Therefore, efficient preprocessing can

be said to be important not just for extractive but also for abstractive summarizing techniques. This module standardizes and refines the text, thus heavily influencing the accuracy and relevance of the summaries generated and hence making the whole system effective.

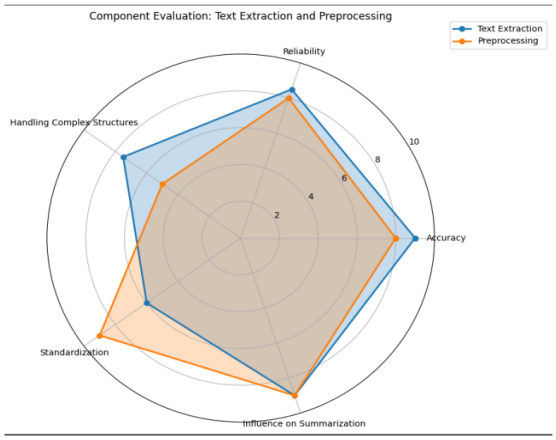

Figure 3.8: Component Evaluation

3.5 Implementation

```
AI_Summarizer/
│
├── src/
│   ├── text_extractor.py
│   ├── fetch_article.py
│   ├── preprocess.py
│   ├── extractive_summarization.py
│   └── abstractive_summarization.py
│
├── templates/
│   └── Index1.html/
│
├── Resources/
│   ├── Install_Resources.bat
│   └── Resources.txt
│
├── App.py
│
├── Run.bat
│
├── Summarizer_App.exe
│
└── README.md
```

Figure 3.9: Implementation of the Web App

Chapter 4

Final Discussion

4.1 Strengths

The system excels in several key areas.

Versatility:

It values versatility by providing both abstractive and extractive summarization approaches. This will help the user generate output that meets his needs, be it a high-level view or detailed extraction of key points. Abstractive summaries are great whenever a user wants a condensed version of the text, reworded; extractive summaries are perfect for users directly quoting and extracting exact excerpts. For this reason, it makes it very appropriate for uses ranging from rapid executive summaries to detailed research analysis.

Scalability:

This tool is designed to scale, ensuring that increasing volumes of data can be managed efficiently without a loss of performance. The back-end infrastructure is strong and able to handle large data sets since it is supported by advanced text processing techniques and optimized data handling techniques. The system remains responsive and effective with increasing user demand and larger data inputs, making it very well-suited for environments with sizable data processing requirements. This scalability provides a guarantee for long-term viability and adaptability in dynamic, data-rich environments.

User Experience:

One of the major strengths of this system lies in user experience. Essentially, it has been designed with simplicity and intuitiveness in mind. The interface is so well designed that loading the documents, choosing the summarization modes, and then viewing the results are not laborious tasks. Little training and technical know-how will be required to use this design since it follows the user-oriented approach—and hence,

capture a larger market audience. If the system manages to engage its users better by making its use easier, then it creates productive and gratifying interaction that eventually strengthens general usability and user satisfaction.

4.2 Limitations

- Dependency on External APIs: Abstractive summarization is dependent on an external API, that is, the Google Gemini API, opening up a range of potential issues with API limits and downtime.

4.3 Implications for the Business Problem

- This solution effectively serves to help meet this growing demand for quick and efficient content summarization. The solution increases engagement and satisfaction by providing the user with concise, relevant summaries, lessening cognitive load on users and allowing fast access to the main information, hence enabling enjoyment and productivity.
- It cuts down the cost of summarization very drastically compared to when such work is done manually. Additional efficiency in this regard aids in cutting down operational costs and, at the same time, makes processing larger data volumes easier for any company. This thus leaves such a company better positioned with respect to scaling its operations to higher demand rates without giving away too much on quality.

GitHub Repository

The full implementation of this project, along with documentation and instructions for setup, can be found in the following GitHub repository:

https://github.com/ms-usmani/B198$_{S}ummarization-Tool-Using-NLP-and-AI$