Intelligent Research Topic Analysis & Agentic AI Research Assistant

Milestone 1: Traditional NLP Research Analysis System

Mid-Semester Project Report
Generative AI Course

Krishiv Saumya Kumar Harshvardhan Gupta

github.com/Harsh-gupta-07/GenAI-Mid-Sem

February 27, 2026

Abstract

This report presents the design and implementation of an intelligent research topic analysis system developed as the first milestone of a two-part Generative AI course project. The system employs classical Natural Language Processing (NLP) techniques to automatically analyze research documents and extract structured insights without relying on Large Language Models (LLMs). Given a research topic or uploaded document, the system performs text preprocessing, keyword extraction via TF-IDF, topic modeling using Latent Dirichlet Allocation (LDA), and extractive summarization using the TextRank algorithm. The application is built with Streamlit and deployed publicly on Streamlit Community Cloud, satisfying the mandatory hosted deployment requirement. This milestone establishes a classical NLP baseline that will be extended into a fully autonomous agentic AI research assistant in Milestone 2.

Contents

1 Introduction

The exponential growth in research literature across all domains has created a significant challenge: researchers and students struggle to efficiently locate, process, and synthesize relevant information from large collections of documents. Manual reading and note-taking are time-consuming and do not scale well. There is a clear need for automated tools that can analyze text at scale and surface the most relevant themes, keywords, and summaries.

This project, *Intelligent Research Topic Analysis & Agentic AI Research Assistant*, addresses this challenge through a two-milestone progression. In **Milestone 1** (this report), we build a traditional NLP-based system capable of processing research documents and producing structured analytical output using classical machine learning and NLP methods. In **Milestone 2** (end-semester), the system will be extended into a fully agentic AI assistant powered by LangGraph, capable of autonomous web retrieval and multi-source reasoning.

The objectives of Milestone 1 are:

- Accept research topics or uploaded documents (PDF, DOCX, TXT) as input.
- Perform standard text preprocessing: tokenization, stop-word removal, and lemmatization.
- Extract the most significant keywords using TF-IDF weighting.
- Identify latent topic clusters using LDA topic modeling.
- Produce an extractive summary using the TextRank algorithm.
- Present results through an interactive, publicly accessible web interface.

1.1 Scope and Constraints

The system was built under the following constraints imposed by the course:

- No LLMs or agentic frameworks are permitted in Milestone 1.
- All libraries must be open-source (NLTK, spaCy, Gensim, scikit-learn).
- The application must be deployed on a public hosting platform (Streamlit Community Cloud).
- Localhost-only demonstrations are not accepted.
- The team consists of three members.

2 Related Work

Automatic text analysis has a long history in computational linguistics and information retrieval. The key techniques employed in this project are grounded in well-established literature.

TF-IDF (Term Frequency-Inverse Document Frequency) is a simple but effective method for ranking the importance of terms in a document relative to a corpus. Despite its simplicity, TF-IDF remains competitive for keyword extraction in single-document and small-corpus scenarios.

Latent Dirichlet Allocation (LDA) is a generative probabilistic model that treats documents as mixtures of topics, where each topic is a distribution over words. LDA has

become the standard approach for unsupervised topic discovery in text corpora.

TextRank adapts the PageRank graph algorithm to natural language. Sentences are represented as nodes in a graph, with edge weights reflecting inter-sentence similarity. The highest-scoring sentences form the extractive summary. TextRank remains a widely used baseline for extractive summarization due to its simplicity and effectiveness without requiring training data.

Wikipedia has been widely used as a knowledge base for NLP research, providing structured, broad-coverage text that can serve as a proxy for domain knowledge when no specific documents are available.

3 System Architecture

3.1 High-Level Architecture

The system follows a modular pipeline architecture, illustrated in Figure 1. The main components are:

1. **Input Layer**: Handles user input either as free-text topic queries (fetched from Wikipedia) or as uploaded document files (PDF, DOCX, TXT).
2. **Data Fetching Module** (`data_fetcher.py`): Extracts raw text from uploaded files or retrieves Wikipedia article content using the `wikipedia` Python package.
3. **NLP Pipeline Module** (`nlp_pipeline.py`): Executes the full analysis pipeline: preprocessing → TF-IDF → LDA → TextRank.
4. **Presentation Layer** (`app.py`): Renders results in the Streamlit UI with keyword visualizations, topic listings, and summaries.

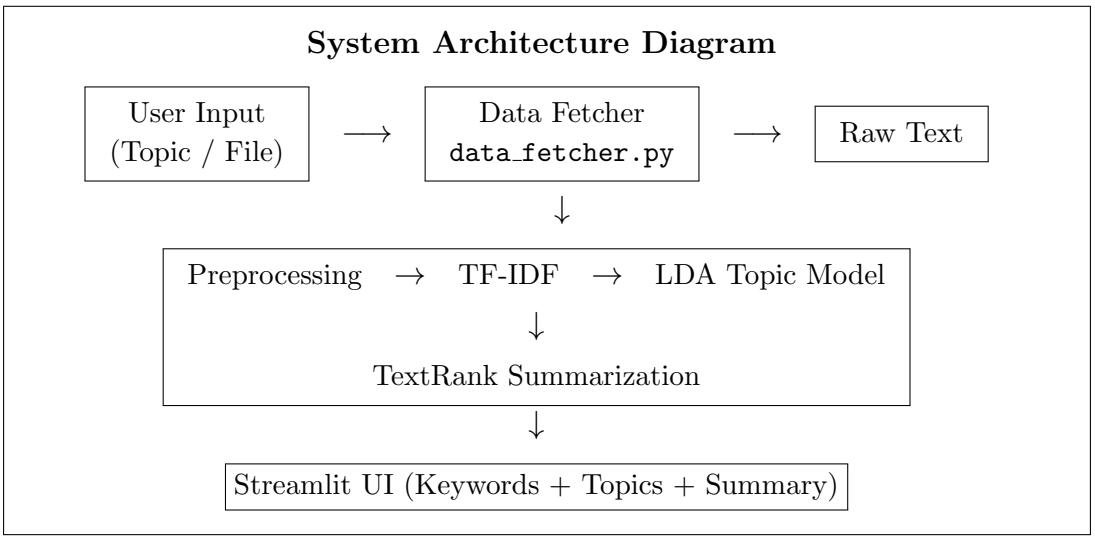

Figure 1: High-level system architecture of the NLP Research Analysis System.

3.2 Project Structure

The repository is organized as follows:

```
GenAI-Mid-Sem/
|-- app.py                # Main Streamlit application
|-- requirements.txt      # Python dependencies
|-- .gitignore
|-- README.md
|-- src/
    |-- data_fetcher.py   # Document extraction & Wikipedia fetching
    |-- nlp_pipeline.py   # NLP processing (preprocessing, TF-IDF, LDA,
        TextRank)
```

Listing 1: Project directory structure

4 Methodology

4.1 Input Handling and Text Extraction

The `data_fetcher.py` module handles two distinct input modes:

- **Document Upload**: Users can upload files in PDF, DOCX, or plain-text (TXT) format. The module uses `PyMuPDF` (for PDF), `python-docx` (for DOCX), and standard file I/O (for TXT) to extract raw text content.
- **Wikipedia Search**: Users enter a topic keyword. The module calls the `wikipedia` Python package to fetch the corresponding article's full text, which is then passed to the NLP pipeline.

4.2 Text Preprocessing

Preprocessing is a critical step to normalize raw text before feature extraction. The pipeline, implemented in `nlp_pipeline.py`, performs the following steps in order:

1. **Tokenization**: The raw text is split into individual word tokens using NLTK's `word_tokenize`.
2. **Lowercasing and Punctuation Removal**: All tokens are lowercased and non-alphabetic tokens are discarded.
3. **Stop-word Removal**: Common English stop-words (e.g., "the", "is", "and") are removed using NLTK's `stopwords` corpus, as they carry no semantic weight.
4. **Lemmatization**: Words are reduced to their dictionary base form using NLTK's `WordNetLemmatizer` (e.g., "running" $\rightarrow$ "run", "studies" $\rightarrow$ "study").

The result is a clean list of semantically meaningful tokens ready for downstream processing.

4.3 Keyword Extraction via TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is used to rank keywords by their relative importance. For a term t in document d from corpus D, the TF-IDF score is defined as:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t, D) \tag{1}$$

where:

$$\text{TF}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}, \quad \text{IDF}(t,D) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right) \tag{2}$$

In this system, the input document is treated as the corpus (split into sentences), and `scikit-learn`'s `TfidfVectorizer` is used to compute scores. The top-N terms by average TF-IDF weight are returned as extracted keywords. A bar chart visualization of keyword scores is generated using `matplotlib`.

4.4 Topic Modeling via LDA

Latent Dirichlet Allocation is applied to discover latent thematic structure in the document. LDA models each document as a mixture of K topics, where each topic is characterized by a distribution over vocabulary terms.

The `Gensim` library is used for LDA. The preprocessing pipeline produces a bag-of-words corpus and a dictionary mapping. Key hyperparameters include:

Table 1: LDA Hyperparameters used in the system

Parameter	**Value**	**Description**
`num_topics`	3 (default)	Number of latent topics to discover
`passes`	10	Number of training passes over the corpus
`alpha`	`auto`	Document-topic density (learned)
`random_state`	42	For reproducibility

Each discovered topic is presented as its top representative keywords, allowing users to interpret the thematic clusters present in the document.

4.5 Extractive Summarization via TextRank

TextRank is a graph-based ranking algorithm adapted for extractive summarization. The algorithm proceeds as follows:

1. The document is split into sentences.
2. Each sentence is represented as a TF-IDF vector.
3. A similarity matrix S is computed where S_{ij} is the cosine similarity between sentences i and j.
4. The similarity matrix is treated as a weighted adjacency matrix of a graph, and the PageRank algorithm is run to score each sentence by its centrality.
5. The top-k highest-scoring sentences are selected and returned in their original order as the summary.

This approach requires no training data and produces summaries that reflect the most interconnected, information-dense content in the document.

5 Implementation Details

5.1 Technology Stack

Table 2 summarizes the libraries and tools used in this milestone.

Table 2: Technology stack for Milestone 1

Component	Library / Tool	Purpose
Frontend UI	Streamlit	Interactive web application
Text Preprocessing	NLTK	Tokenization, stop-words, lemmat
Keyword Extraction	scikit-learn (`TfidfVectorizer`)	TF-IDF computation
Topic Modeling	Gensim (`LdaModel`)	LDA topic discovery
Summarization	Custom TextRank (scikit-learn + NumPy)	Extractive summarization
PDF Parsing	PyMuPDF (`fitz`)	Extract text from PDF files
DOCX Parsing	python-docx	Extract text from Word documen
Wikipedia Retrieval	`wikipedia` Python package	Fetch topic content
Visualization	matplotlib	Keyword bar chart
Deployment	Streamlit Community Cloud	Public hosting

5.2 Key Code Components

The NLP pipeline is encapsulated in `nlp_pipeline.py`. The main functions are:

```
def preprocess_text(text: str) -> list[str]:
    """Tokenize, remove stop-words, and lemmatize input text."""

def extract_keywords_tfidf(text: str, top_n: int = 15) -> dict:
    """Return top-N keywords with their TF-IDF scores."""

def perform_lda(text: str, num_topics: int = 3) -> list[tuple]:
    """Run LDA and return list of (topic_id, top_words) tuples."""

def textrank_summarize(text: str, num_sentences: int = 5) -> str:
    """Return extractive summary using TextRank algorithm."""
```

Listing 2: Core pipeline function signatures

5.3 Deployment

The application is deployed on **Streamlit Community Cloud** and is publicly accessible. This satisfies the mandatory hosting requirement of the course. The deployment uses the `app.py` entry point with dependencies specified in `requirements.txt`.

Live URL: harsh-gupta-07-genai-mid-sem-app-ymrqvk.streamlit.app

6 Results and Evaluation

6.1 Functional Output Description

The system produces three main types of output for any given input:

1. **Keyword Extraction**: A ranked list of the top 15 TF-IDF keywords, displayed as a horizontal bar chart with scores. This gives an immediate sense of the most important vocabulary in the document.
2. **Topic Clusters**: Three LDA-discovered topic groups, each represented by its top 5-8 associated words. These allow the user to identify sub-themes within the document.
3. **Extractive Summary**: A coherent 3-5 sentence summary composed of the highest-ranked TextRank sentences, preserving the original sentence structure and meaning.

6.2 Sample Test Cases

We tested the system on the following input types:

Table 3: Test cases and observed behaviour

Input	Input Type	Observed Output Quality
"Machine Learning"	Wikipedia topic query	Relevant keywords (e.g., *model, training, data*); 3 coherent topics covering supervised, unsupervised, and neural approaches; accurate extractive summary.
Research paper (PDF)	Uploaded PDF document	Domain-specific keywords extracted correctly; topic clusters reflect paper sections; summary captures abstract-level content.
Short TXT article	Uploaded TXT file	Keywords and summary quality slightly reduced due to smaller text size; LDA topics less distinct with limited input.

6.3 Limitations

- **Short Documents**: LDA topic modeling performs poorly on very short texts (under $\sim$200 words) as there is insufficient data to learn meaningful topic distributions.
- **Domain Specificity**: Stop-word lists and preprocessing are tuned for general English; highly technical or domain-specific jargon may not be handled optimally.
- **Language Support**: The current system supports English text only; multi-lingual documents will not produce meaningful results.
- **No Ground-Truth Evaluation**: Without labelled benchmark datasets, quantitative evaluation metrics (e.g., ROUGE for summarization, topic coherence for LDA) were computed informally. Adding formal evaluation is a target for future work.

7 Input-Output Specification

Table 4 provides the formal input-output specification for the system.

Table 4: Input-output specification

Mode	Input	Output
Topic Search	Free-text research topic string (e.g., "Quantum Computing")	Wikipedia article fetched automatically; processed and analyzed.
Document Upload	File in PDF, DOCX, or TXT format (English text)	Text extracted and passed to NLP pipeline.
Common Outputs for both modes:		Top-N keywords with TF-IDF scores (bar chart) K topic clusters (word lists) Extractive summary (3-5 sentences)

8 Milestone 2 Roadmap

Milestone 2 will transform this traditional NLP system into a fully autonomous AI research agent. The planned extensions include:

- **LangGraph Integration**: Implementing stateful agentic workflows where each step (retrieve, reason, synthesize) is a node in a directed graph.
- **Web Search & Retrieval**: Integrating Tavily or DuckDuckGo search APIs to retrieve live, up-to-date information from the web.
- **LLM-Powered Reasoning**: Replacing classical NLP components with open-source LLMs (via Hugging Face or Ollama) for abstractive summarization and multi-document reasoning.
- **Structured Report Generation**: Automatically generating formatted research reports from aggregated multi-source content.
- **State Management**: Maintaining conversation and research state across multiple agent steps using LangGraph's state graph.

9 Conclusion

This report presented the design and implementation of an NLP-based research document analysis system (Milestone 1 of the Intelligent Research Assistant project). The system successfully fulfills all Milestone 1 requirements: it accepts research topics or uploaded documents, performs text preprocessing, extracts keywords via TF-IDF, discovers latent topics via LDA, and produces extractive summaries via TextRank — all presented through a deployed, publicly accessible Streamlit web application.

The implementation demonstrates a clear understanding of classical NLP and information retrieval techniques and provides a strong functional baseline for the agentic system to be developed in Milestone 2. The modular architecture of the codebase (separating data fetching, NLP logic, and UI) will facilitate seamless integration of LangGraph and LLM-based components in the next phase.