Group 7: LLMotive

Project Title: Summarization of a Stack of Papers using LLMs (SSPL) – Stage 2

Arianna Matienzo

Website Designer

Atharva Dalvi

System Architect

Khushi Nankani

Data Reporting Analyst

Olu Ogunnirian

Marketing Designer

*Abstract* — As academic literature continues to expand, keeping up with key insights becomes increasingly challenging. Traditional summarization methods are often time-consuming and labor-intensive. This project explores how Large Language Models (LLMs) can automate the process in a three-step approach: first, summarizing individual papers, then categorizing them by topic, and finally combining the summaries into a cohesive document. By leveraging LLMs’ natural language processing capabilities, the system aims to improve literature review efficiency, allowing researchers and professionals to quickly grasp essential insights without sacrificing accuracy or detail. Early-stage prototypes have demonstrated a functional system that processes papers at scale, and ongoing improvements focus on enhancing summarization speed and handling domain-specific technical language.

*Keywords:* ***Summarization, LLMs, NLP, Automated Literature Review, Research Synthesis, Prototype, Efficiency***

# DESCRIPTION

As the volume of academic literature continues to expand, researchers face increasing challenges in extracting key insights efficiently. Traditional summarization methods require significant time and effort, often leading to information overload. Our project leverages Large Language Models (LLMs) to automate the summarization of research papers, making literature reviews more efficient and accessible. The system follows a structured three-stage process:

1. **Topic Recognition** – NLP techniques categorize research papers based on themes and subject matter.
2. **Summarization** – The system generates concise, meaningful summaries while preserving critical technical details.
3. **Collation** – The summarized content is organized into a cohesive document for easy accessibility.

# MOTIVATION

Our solution enhances research productivity by reducing manual effort, improving knowledge retention, and enabling users to focus on analysis rather than data extraction. The exponential growth of research publications presents a significant challenge in staying updated with new findings. By implementing an automated summarization system, we aim to:

* Save researchers time by providing well-structured summaries.
* Minimize the risk of information overload.
* Improve accessibility to knowledge across disciplines.
* Enhance cross-referencing and synthesis of research findings.

1. **SYSTEM ANALYSIS AND DECOMPOSITION**

**Use Case Diagram:** The use case diagram outlines the interaction between the user and our **Summarization System**. The process begins with the user uploading PDFs, after which the system extracts and cleans the text. It then chunks the content for parallel summarization, performs topic modeling to identify key themes, and generates a final summary for download. This structured workflow ensures efficient and accurate summarization of research papers.

A diagram of a summarization system

AI-generated content may be incorrect.

Image: Use Case Diagram

**Sequence Diagram:** The sequence diagram outlines the step-by-step interactions for efficient document summarization using parallel processing.

1. User starts the process.
2. Main Program loads PDFs and extracts text via the PDF Processor.
3. Text Cleaner processes the text, and Text Chunker splits it into smaller sections.
4. Summarizer processes chunks in parallel, and the Topic Modeler organizes them by topic.
5. Parallel Executor manages execution, and Output Writer stores the final summaries.
6. User receives the summaries.

A diagram of a project

AI-generated content may be incorrect.

Image: Sequence Diagram

**Class Diagram:** The diagram shows how four classes work together:

1. MainController starts the program and uses TopicModeler.
2. TopicModeler loads papers and analyzes topics, using Summarizer to summarize content.
3. Summarizer breaks text into chunks and summarizes them.
4. PDFProcessor extracts and cleans text from PDFs.

Together, they process PDFs, model topics, and summarize the content.

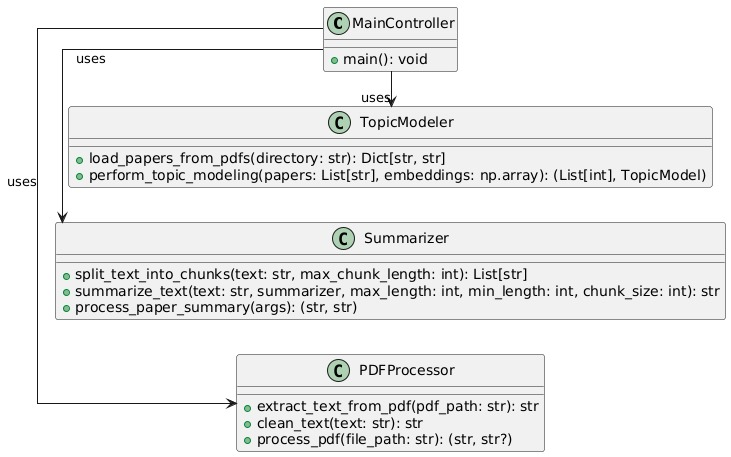


Image: Class Diagram

1. **POTENTIAL SOLUTIONS AND SYSTEM ARCHITETURE**

**1. System Architecture**

We propose a modular system with the following architecture:

* **Data Input Layer**: Accepts PDF/Word documents for processing.
* **Processing Layer**: Includes NLP techniques for topic classification and summarization.
* **Storage Layer**: Stores categorized and summarized data for future retrieval.
* **Presentation Layer**: Provides an intuitive interface for user interaction.

**2. Algorithm and Flowcharts**

**Architecture Diagram:** This architecture diagram shows the process flow for handling PDF documents:

1. User: Uploads PDFs.
2. PDF Processing: Extracts and cleans text from PDFs.
3. Topic Modeling and Making Graphs: Generates text embeddings, clusters topics, assigns papers to topics, and creates topic graphs.
4. Summarization: Splits text into chunks and summarizes it.
5. Execution: Uses parallel processing to execute summarization.
6. Output: Generates a summary report.

Together, these steps process PDFs, model topics, and summarize the content efficiently.

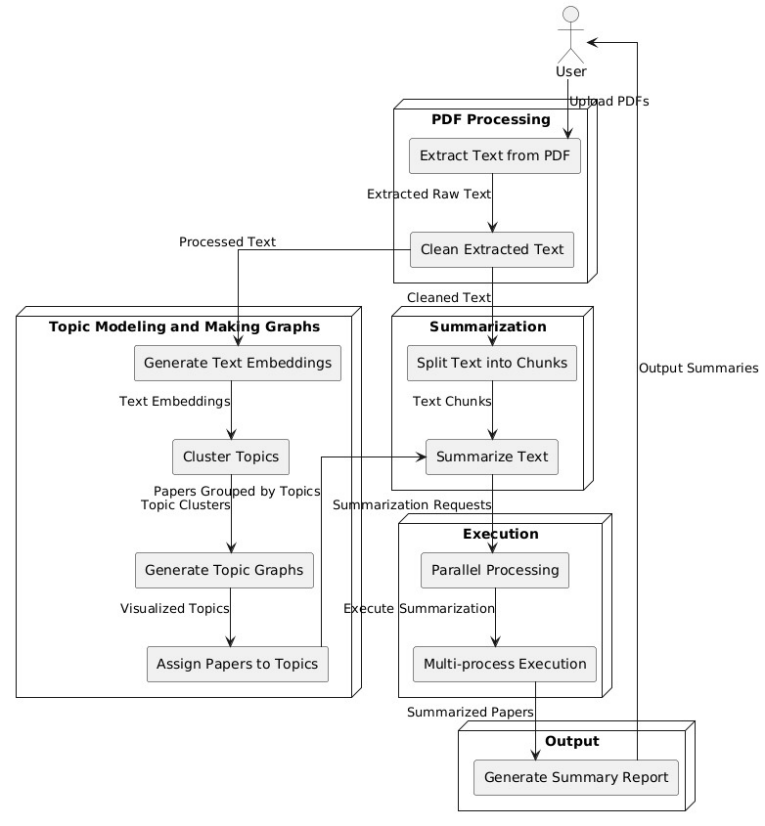


Image: Architecture Diagram

**Algorithm:** Algorithm for PDF Extraction & Cleaning

* Extract text using PyPDF2.
* Remove noise
* Return clean text if it has enough words.

Algorithm for Chunking Strategy

* **Tokenize text into sentences (nltk.sent\_tokenize).**
* **Create text chunks up to a maximum length**
* **Return chunks for processing.**

Algorithm for Parallel Summarization

1. Use ThreadPoolExecutor for I/O-bound tasks (PDF processing).
2. Use ProcessPoolExecutor for CPU-bound tasks (summarization).
3. Summarize text in parallel chunks.

**Flowchart:** This flowchart outlines the process of handling PDFs:

1. Load PDFs from Directory.
2. Extract & Clean Text from the PDFs.
3. Split into Chunks for easier processing.
4. Decision Point: If there are fewer than 2 chunks:
   * Yes: Process sequentially.
   * No: Use ThreadPoolExecutor for parallel processing.
5. Summarize Chunks to condense the information.
6. Merge Summaries into a cohesive document.
7. Save to Output File for final use.

This flowchart ensures efficient and organized processing of PDF documents.

A diagram of a process

AI-generated content may be incorrect.

Image: Flowchart

1. **IMPLEMENTATION PLAN**

**1. Testbed**

We will use a dataset of **50+ research papers** related to LLM applications in semiconductors. The system will be tested for accuracy, processing time, and scalability.

**2. Programming Language & Tools**

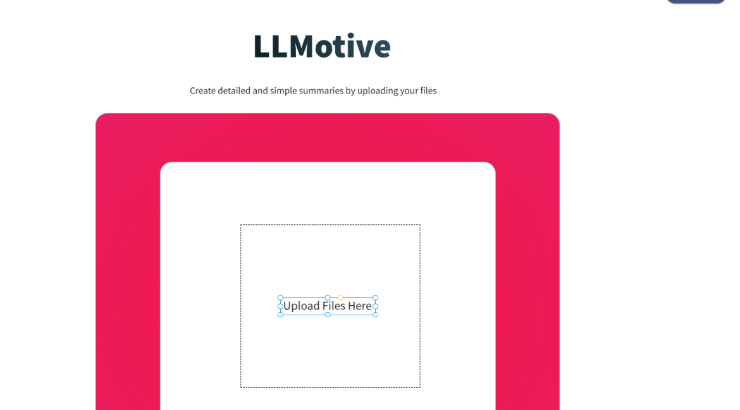
* **Programming Language**: Python (for NLP processing and LLM integration)
* **Development Frameworks**:
  + Hugging Face Transformers (for LLMs)
  + TensorFlow/PyTorch (for deep learning-based models)
  + Scikit-learn (for clustering and topic modeling)
* **Deployment & UI**: Flask/Django for web-based access

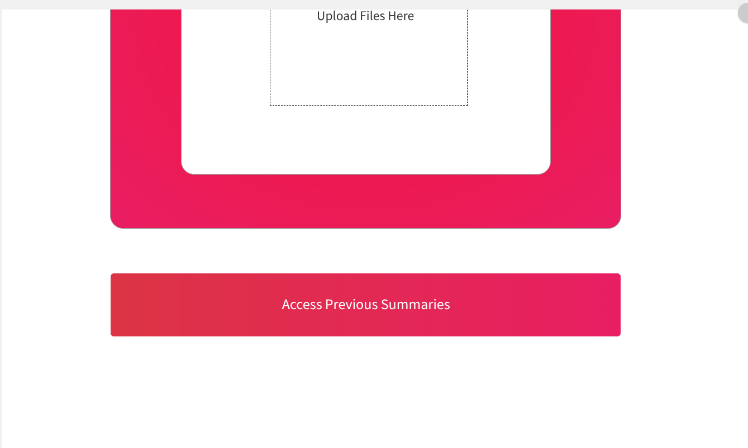
1. **Development Devices**

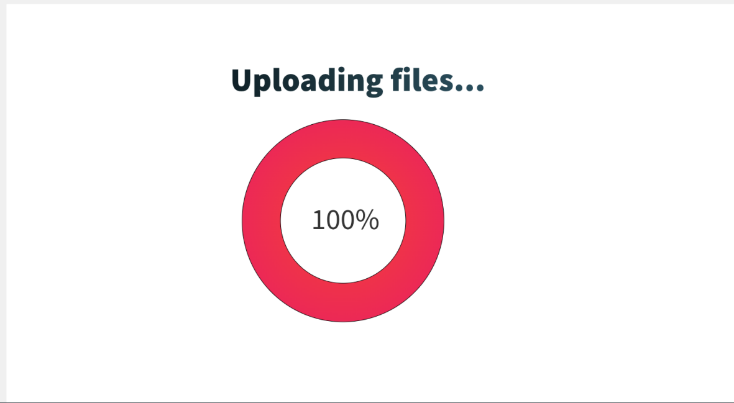
* High-performance GPU-enabled machines (e.g., NVIDIA RTX 3090)
* Cloud-based AI services (e.g., Google Colab, AWS, or Azure AI)

# **INITIAL VERSION OF USER INTERFACE**

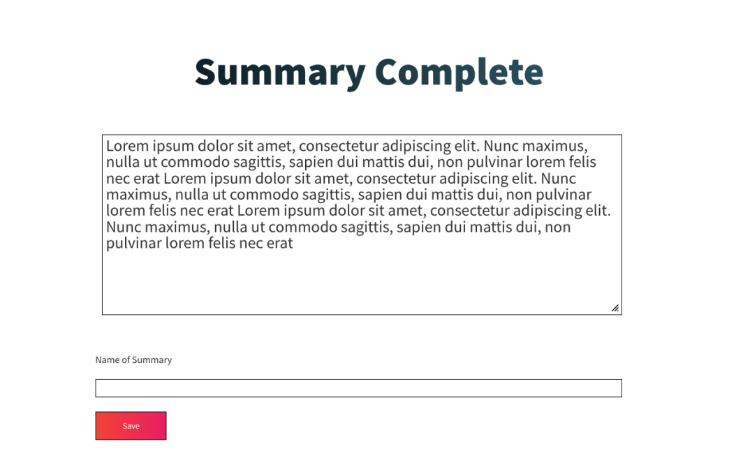
Upload Files Feature:







Review and Save Generated Summary:



Access Previous Summaries



# **NEXT STEPS**

* Develop the final front-end interface for user interaction.
* Develop the back end to generate quality summaries in effective time.
* Conduct performance benchmarking to refine processing time and scalability.