AI RESEARCH PAPER ASSISTANT

Team 5

**Team Lead**: T. Sathvika [22B21A4311]

**Teammates**

K. Sai Kiran [22B21A4349]

G. Durga Bhavani [22B21A4308]

A. Anand Charan [22B21A4382]

J.V. Sai Ramakumar [22B21A4383]

Department: CSE-AI

Institution: Kakinada institute of engineering and technology

Academic year: 2025

**Abstract**

Reading and analyzing research papers can be slow and challenging. This is especially true for students and young researchers who need to review many papers in a short amount of time. The AI Research Paper Assistant was created to help make this easier using Natural Language Processing (NLP) techniques. The system can automatically summarize long research papers, extract key keywords, and answer specific questions from the paper’s content.

The solution uses transformer-based models: DistilBART for generating clear, short summaries, KeyBERT for finding the most relevant keywords, and DistilBERT for answering questions. The workflow starts with uploading a PDF, then moves on to text extraction and cleaning. After preprocessing, the models analyze the content and produce results that appear in a React-based user interface. The backend runs on Python with Flask or FastAPI.

This tool aims to save time, improve understanding, and support academic work by providing students and researchers with a straightforward, effective way to work with complex papers.

# Introduction & Problem Statement

**Background of the Problem**

In today’s academic world, research plays a crucial role in driving innovation and development. Every month, thousands of papers are published across diverse fields such as computer science, engineering, medicine, and social sciences. While this continuous flow of knowledge is valuable, it creates a challenge for students, teachers, and researchers who need to stay updated. Reading and analyzing research papers demands significant time and focus because most papers are lengthy, highly technical, and filled with specialized terminology. Traditional approaches like skimming or highlighting important sections often require effort and may still cause readers to miss critical insights.

**Why It Is Important**

The difficulty of engaging with large volumes of research becomes more serious when students and researchers must review multiple papers, prepare for exams, conduct literature surveys, or compare findings for projects. Without an efficient method, they risk overlooking important studies or being overwhelmed by excessive information. This not only affects academic progress but can also slow down innovation and decision-making in real-world applications. A system that can reduce reading time while preserving the essence of research is essential for improving productivity, comprehension, and accessibility of knowledge.

**Clear Definition of the Problem/Objectives**

The main problem is the **time and effort required to process and understand research papers**. The objective of this project is to design an **AI Research Paper Assistant** that addresses this issue by:

1. **Automatic Summarization** – generating concise and clear summaries of lengthy papers.
2. **Keyword Extraction** – identifying and highlighting the most important concepts and terms.
3. **Question Answering** – allowing users to directly ask questions and receive specific answers from the paper’s content.

By achieving these objectives, the project aims to save time, improve comprehension, and provide a more interactive way of learning, ultimately making research more accessible to students, teachers, and researchers.

Proposed Methodology

The proposed system follows a **step-by-step Machine Learning (ML) pipeline**, ensuring that the AI Research Paper Assistant functions in a structured and reliable manner. The methodology includes **data gathering, preprocessing, feature extraction, model integration, and results presentation**.

**1. Data Gathering**

The system mainly relies on **user-uploaded research papers in PDF format** as its primary data source. These papers can come from platforms such as **arXiv, IEEE Xplore, Springer, or Google Scholar**. Since the assistant is designed to process uploaded files on demand, it does not strictly require a fixed dataset.

However, for research or fine-tuning purposes, publicly available datasets can be used, such as:

* **arXiv Summarization Dataset** – a large collection of scientific papers with summaries.
* **SciTLDR** – a dataset containing short, human-written summaries of scientific articles.
* **(Stanford Question Answering Dataset)** – widely used for training and evaluating question-answering models.

Including these datasets allows the system to be fine-tuned for better performance on academic text.

**2. Data Ingestion and Preprocessing**

Users upload research papers in PDF format. The raw PDF content is extracted using libraries like **PyPDF2** or **pdfplumber**. Since extracted text often contains noise such as line breaks, headers, and unwanted symbols, preprocessing ensures the content is cleaned and tokenized into meaningful units. This step prepares the text for further Natural Language Processing (NLP) tasks.

**3. Feature Extraction**

After preprocessing, the text is transformed into structured features:

* **TF-IDF (Term Frequency–Inverse Document Frequency)** for capturing important terms.
* **Text Embeddings** for representing words and sentences in semantic space, ensuring contextual meaning is preserved.

These features act as the foundation for downstream tasks.

**4. Model Integration**

The system uses pre-trained transformer models from Hugging Face:

* **DistilBART** for summarization, reducing lengthy paragraphs into concise summaries.
* **KeyBERT** for keyword extraction, identifying the most relevant terms.
* **DistilBERT** for Q&A, enabling users to query the paper directly.

The backend is built in **Python** using **Flask** or **FastAPI**. Preprocessing and feature engineering use **scikit-learn, NumPy, and Pandas**, while **Matplotlib** may be used for visualization. The frontend is developed using **ReactJS** or **basic HTML/CSS/JavaScript**, providing an interactive user interface.

* **Scratch Developing Model**

**1. Collect Data**

* **Sources: Collect a large set of research papers from platforms like arXiv, IEEE Xplore, Springer, and Google Scholar. These provide a diverse range of scientific content.**
* **Preprocessing:**
  + **Download papers in PDF format.**
  + **Use tools such as PDFMiner, PyMuPDF, or GROBID to convert PDFs into clean plain text.**
  + **Remove unwanted sections like references, figure captions, and formatting artifacts.**
* **Labeling: Create small but high-quality labeled datasets:**
  + **Summaries → Human-written short versions of papers (abstracts or manually curated).**
  + **Question–Answer Pairs → Generate Q&A from text passages. Example: input = paragraph, output = relevant question + answer.**
  + **Keywords → Extract key terms or phrases that represent the core of each paper. These can be manually tagged or semi-automatically extracted.**

**2. Build Tokenizer**

* **A tokenizer is a tool that breaks text into smaller units (subwords/wordpieces) that the model can process.**
* **Instead of using a pre-trained tokenizer, train your own tokenizer on the collected dataset so it learns vocabulary specific to research papers.**
* **Recommended approaches: Byte-Pair Encoding (BPE), WordPiece, or SentencePiece.**
* **Output: A custom vocabulary file that defines how input text is split into tokens. This ensures the model understands scientific terms, equations, and abbreviations better.**

**3. Design Model**

* **Use simplified Transformer architectures to reduce computational cost.**
* **Build different models for different tasks:**
  + **Summarizer (Encoder–Decoder model) → Input: full text → Output: short summary.**
  + **Q&A Model (Encoder-only, like BERT-style) → Input: question + passage → Output: answer span or text.**
  + **Keyword Extractor (Encoder with ranking head) → Input: text → Output: ranked list of important keywords.**
* **Keep models lightweight (few layers, smaller hidden dimensions) so they can be trained on limited hardware.**

**4. Train From Scratch**

* **Initialize the model with random weights (no transfer learning).**
* **Training strategy:**
  + **Use your labeled datasets for each task.**
  + **Train in mini-batches using frameworks like PyTorch or TensorFlow.**
  + **Apply optimizers such as AdamW with appropriate learning rates.**
* **Hyperparameters to tune:**
  + **Learning rate**
  + **Batch size**
  + **Number of epochs**
  + **Dropout rate**
* **Expect slower convergence than pre-trained models; training may require multiple attempts.**

**5. Evaluate**

* **Once trained, evaluate models on validation datasets.**
* **Summarization:**
  + **Use ROUGE scores (ROUGE-1, ROUGE-2, ROUGE-L) to measure overlap with human-written summaries.**
  + **Human evaluation is also valuable for fluency and readability.**
* **Q&A:**
  + **Use Accuracy, Precision, Recall, and F1 Score to measure how well answers match ground truth.**
* **Keyword Extraction:**
  + **Measure Precision, Recall, and F1 Score against manually labeled keywords.**
* **Iteratively improve the model by adjusting dataset size, training duration, or architecture.**

**6. Deploy**

* **Once the models are trained and evaluated, deploy them for real usage.**
* **Export/Save: Store trained models and tokenizer in a structured format (e.g., PyTorch .pt or TensorFlow .h5).**
* **Backend: Use frameworks like Flask or FastAPI to serve models through APIs. Example:**
  + **/summarize → Returns summary of input text.**
  + **/qa → Returns answer for a given question and context.**
  + **/keywords → Returns extracted keywords.**
* **User Interface: You can build a simple web app or dashboard where users can upload papers, ask questions, and view outputs.**

**Tools / Libraries**

* Hugging Face Transformers (DistilBART, DistilBERT)
* KeyBERT
* scikit-learn
* NLTK / spaCy
* PyPDF2 / pdfplumber
* Flask / FastAPI
* ReactJS / HTML / CSS / JavaScript
* LangChain (optional)
* NumPy / Pandas

**5. Expected System Workflow**

**PDF Upload → Text Extraction & Cleaning → Feature Extraction → Model Inference → Results Display**

At inference, three modules (summarization, keyword extraction, Q&A) work in parallel and present results to the user.

**6. Block Diagram**

PDF Upload

↓

Text Extraction & Cleaning

↓

Feature Extraction (Embeddings)

↓

┌────────────┬───────────────┬────────────┐

│ Summarization │ Keyword Extraction │ Q&A │

└───────┬──────┴──────────┬─────────┴─────┘

↓

Results Displayed in Web UI

**7. Expected Working**

When a user uploads a paper, the assistant automatically extracts, preprocesses, and analyzes the text. Summaries, important keywords, and answers to specific user queries are generated and displayed in the interface. This workflow ensures that the system saves time while improving comprehension of research material.

**Conclusion**

The main goal of the AI Research Paper Assistant's development was to make it easier to comprehend and evaluate scholarly research papers. The system's three main functions—automatic summarization, keyword extraction, and question answering—are achieved by combining Natural Language Processing (NLP) methods with transformer models that have already been trained. By concentrating on the most crucial portions of a paper instead of reading it word for word, these features help users save time and effort.

There are many uses for this tool. It can be a useful tool for students to use when working on projects or creating literature reviews. It can give educators and assessors a brief synopsis of papers, facilitating their efficient review of numerous documents. Because it makes it possible to quickly explore and filter vast collections of scholarly articles, it increases researcher productivity.

**References**

1. Hugging Face Transformers. Available at: <https://huggingface.co>
2. KeyBERT: Keyword Extraction with BERT. Available at: <https://github.com/MaartenGr/KeyBERT>
3. FastAPI Framework. Available at: <https://fastapi.tiangolo.com>
4. PyPDF2 Library. Available at: <https://pypi.org/project/PyPDF2/>
5. ReactJS Documentation. Available at: <https://react.dev>
6. arXiv Dataset. A large collection of scientific papers from arXiv used for text summarization and classification tasks. Available at: https://www.kaggle.com/datasets/Cornell-University/arxiv
7. SciTLDR: Extreme Summarization of Scientific Documents. A dataset of scientific papers paired with TLDR-style summaries. Available at: <https://github.com/allenai/scitldr>
8. SQuAD (Stanford Question Answering Dataset). Widely used dataset for training and evaluating question answering models. Available at: https://rajpurkar.github.io/SQuAD-explorer/
9. CORD-19 Dataset. A dataset of scholarly articles about COVID-19 and related research, often used in NLP tasks. Available at: https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge