Digital Egypt Pioneers Initiative - DEPI

مبادرة رواد مصر الرقمية

Graduation Project Documentation

Title of project	**SciBot: Your AI Assistant for Research Paper Comprehension**	
Topic	**Personalized chatbot for dynamic responses**	
Track	**AI& Data Science**	**Generative AI- professional**
Year	2024-2025	
Group Code	ONL2_AIS3_S1	
Instructor	Eng. Mohammed Agoor	

Group number: 1

No.	**Student Name**	**Email**	**Phone Number**
1	Gannatullah Asaad Abdallah	204115@eru.edu.eg	+201092734918
2	Nada Mokhtar	N.Hussain2128@nu.edu.eg	+201030511783
3	Abdalla Maged Gamal	abdallaellaithy@gmail.com	+201016568880
4	Alyaa Atef	alyaatef2003@gmail.com	+201033738437
5	Arwa Ibrahim	testfreelance2003@gmail.com	+201028094651
6	Salma Said	salmattaahhaa@gmail.com	+201208181689

Project github link:

https://github.com/GannaAsaad/Personalized-Chatbot-Project.git

Content

Introduction

The growing volume and complexity of academic research can make it difficult for students, researchers, and professionals to quickly grasp the key ideas and contributions of a scientific paper. Many struggle with unfamiliar terminology, dense writing, or the time constraints of staying updated with current literature. To address this challenge, we have developed an AI-powered chatbot that simplifies research paper comprehension and enables interactive exploration of its content.

This chatbot accepts a PDF file of a research paper, automatically extracts and organizes its contents, and provides a clear, human-readable explanation of the work. Beyond summarization, the chatbot is also equipped with a highly capable question-answering system that allows users to ask detailed queries about the paper. It remembers previous interactions within the same session, ensuring a coherent and intelligent dialogue experience.

Project Description

The chatbot project integrates large language models, semantic search, and conversational memory to create an intelligent assistant for exploring research papers. Below is a detailed description of its core features and architecture.

1. Core Functionalities

a. Explanation Module

- **Objective**: Deliver a simplified, section-wise explanation of a research paper.
- **Model Used**: gemini-1.5-flash-latest
- **Process**:
  1. PDF Understanding: The input paper is parsed and text is extracted using reliable PDF processing libraries.
  2. Section Extraction: Key sections such as *Abstract*, *Introduction*, *Methodology*, *Results*, and *Conclusion* are identified and isolated.

3. Explanation: The model processes each section individually and generates simplified, human-readable explanations suitable for non-experts.

b. Question-Answering via RAG (Retrieval-Augmented Generation)

- **Objective**: Allow users to interactively ask questions about the paper and receive accurate, context-aware answers.
- **Embedding Model**: all-MiniLM-L6-v2 (used to convert paper content into vector representations for efficient semantic retrieval)
- **Retrieval Model**: Based on a vector store indexed from the processed document
- **Answering Model**: gemini-1.5-flash-latest, which combines retrieved information with powerful generative capabilities to answer questions reliably.

c. Conversational Memory

- **Implementation**: ConversationalRetrievalChain is used to retain session memory and context during the user-chatbot interaction.
- **Benefits**:
  - Maintains awareness of past questions and answers
  - Enables multi-turn, follow-up discussions
  - Improves coherence and engagement in extended sessions

2. Technical Architecture

COMPONENT	TECHNOLOGY OR MODEL USED
FRONTEND	HTML, CSS, Javascript
BACKEND FRAMEWORK	FastAPI
EXPLANATION MODEL	gemini-1.5-flash-latest
RAG ANSWERING MODEL	gemini-1.5-flash-latest
EMBEDDING MODEL	all-MiniLM-L6-v2
MEMORY MANAGEMENT	ConversationalRetrievalChain (LangChain)
PDF PARSING	Libraries such as PyMuPDF or PDFPlumber
DEPLOYMENT	FastAPI (suitable for local or cloud deployment)

3. Use Cases

This tool is ideal for:

- Students looking to simplify complex academic texts
- Researchers conducting fast literature reviews
- Educators wanting to create accessible learning materials
- Professionals needing to stay informed across disciplines

Data Collection and Preprocessing

SciBot's processing pipeline begins with the ingestion of research papers in PDF format. To serve its dual purpose—explaining the content of scientific papers in simplified language and providing accurate answers to user queries—the system performs a structured and modular preprocessing workflow. This ensures both tasks receive high-quality, context-rich input for downstream modeling.

1. PDF Text Extraction

Two methods are implemented for extracting text, depending on the source and interface:

- **Byte-based Extraction (Explanation Task)**
  Used when the PDF file is uploaded as a byte stream.
  - **Library Used:** `PyPDF2`
  - **Function:** `extract_text_from_pdf_bytes()`
  - **Purpose:** Converts the entire PDF file into a single clean text string.
- **Path-based Extraction (RAG Task)**
  Used when the PDF is available as a file path.
  - **Library Used:** `fitz` from `PyMuPDF`
  - **Function:** `extract_text_from_pdf()`
  - **Purpose:** Extracts page-wise text to support consistent parsing and indexing for retrieval.

Both methods include error handling to manage empty documents, inaccessible files, and OCR limitations.

---

2. Section Identification (Explanation Task)

To deliver accurate, section-specific explanations, the extracted text is parsed using regular expressions to isolate key segments of the paper:

- **Function:** extract_sections(text)
- **Targeted Sections:**

  - Abstract
  - Introduction
  - Methodology / Methods / Materials and Methods
  - Results / Findings
  - Conclusion / Conclusions / Discussion and Conclusion
- **Technique:**
  A combined regex pattern is compiled from SECTION_PATTERNS, and matches are identified across the cleaned full text. Boundaries between matched headers are used to segment the text into logical sections.

---

3. Text Chunking (RAG Task)

For the retrieval-based Q&A system, the extracted full text is split into manageable, semantically meaningful segments:

- **Library Used:** RecursiveCharacterTextSplitter from LangChain
- **Function:** create_vector_store(pdf_text, embeddings_model)
- **Chunk Parameters:**
  - chunk_size: 1000 characters
  - chunk_overlap: 200 characters

This strategy ensures:

- Retrieval accuracy across question boundaries
- Context preservation across adjacent chunks
- Compatibility with long-context models like Gemini

---

4. Vectorization and Indexing (RAG Task)

Once the text is chunked, each segment is converted into an embedding and stored for semantic search:

- **Embedding Model:** all-MiniLM-L6-v2 (via HuggingFace)
- **Vector Store:** FAISS (in-memory indexing for efficient retrieval)

This step powers the chatbot's ability to locate the most relevant document segments when responding to user queries.

Model Development

The development of SciBot integrates multiple state-of-the-art natural language processing components to perform two main tasks: **(1) simplifying research paper sections** and **(2) enabling retrieval-augmented question answering** through a contextual chatbot interface. Each task is powered by specialized models and carefully designed pipelines, described below.

1. Explanation Model Pipeline

Objective

Automatically simplify the key sections of a research paper into clear, accessible summaries that can be understood by non-experts, while preserving the accuracy and intent of the original content.

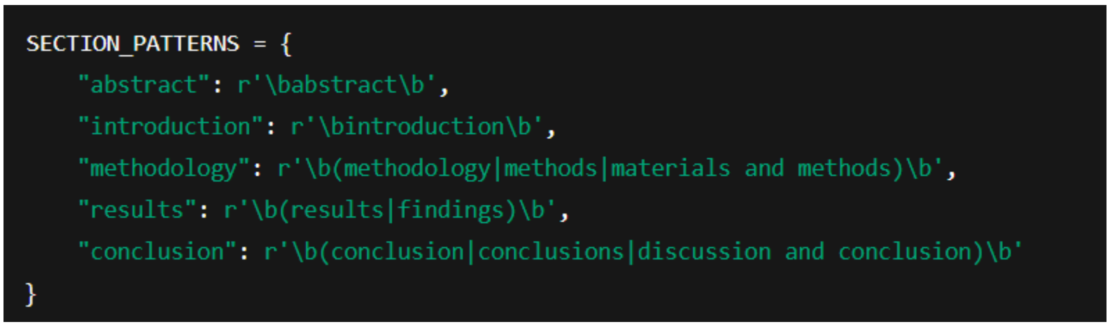
Models & Techniques Used

- **LLM:** gemini-1.5-flash-latest (via generate_content)
- **Prompting Strategy:** Custom structured prompts using LangChain's PromptTemplate
- **Output Formatting:** Structured output enforced as a JSON object using LangChain's StructuredOutputParser

- **Section Matching:** Regular expression-based heuristics to identify and extract standard sections (e.g., *Abstract*, *Introduction*, *Methodology*, *Results*, *Conclusion*) from the full document

Pipeline Steps

1. **PDF Text Extraction**
   - Extract raw text from the uploaded PDF using PyPDF2.
2. **Section Detection**
   - Identify key sections using pre-defined regex patterns:



```
SECTION_PATTERNS = {
    "abstract": r'\babstract\b',
    "introduction": r'\bintroduction\b',
    "methodology": r'\b(methodology|methods|materials and methods)\b',
    "results": r'\b(results|findings)\b',
    "conclusion": r'\b(conclusion|conclusions|discussion and conclusion)\b'
}
```

3. **Prompt Construction**
   - For each extracted section, a prompt is constructed to instruct the model to generate a simplified explanation in plain English, formatted as JSON.
4. **Content Generation**
   - Gemini generates the explanation using the constructed prompt and returns a single-paragraph summary encapsulated within a structured JSON format.
5. **Error Handling and Fallbacks**
   - If the model fails to return a valid JSON object, the raw text is parsed heuristically or fallback explanations are returned.

2. Retrieval-Augmented Generation (RAG) Chatbot

Objective

Enable users to interactively ask questions about the uploaded paper and receive grounded, context-aware answers.

Components

- **Embedding Model:** all-MiniLM-L6-v2 via HuggingFaceEmbeddings (for semantic indexing)
- **Vector Store:** FAISS (efficient in-memory search index for fast similarity search)
- **Answering Model:** gemini-1.5-flash-latest via ChatGoogleGenerativeAI
- **Conversation Framework:** ConversationalRetrievalChain from LangChain
- **Memory System:** ConversationBufferMemory (preserves chat history and context)

Pipeline Steps

1. **PDF Parsing and Chunking**
   - Text is extracted from the uploaded PDF.
   - The full text is split into overlapping semantic chunks using RecursiveCharacterTextSplitter (chunk size: 1000, overlap: 200).
2. **Vector Store Creation**
   - Text chunks are embedded using all-MiniLM-L6-v2 and indexed into a FAISS store.
3. **RAG Construction**
   - At query time, top-k chunks relevant to the user's question are retrieved from FAISS.
   - These are passed along with the user’s query to the Gemini model to generate a response grounded in the retrieved document context.
4. **Conversational Memory**
   - The system maintains memory of prior interactions using ConversationBufferMemory, enabling it to handle follow-up questions naturally.

Advanced Techniques

SciBot incorporates several advanced techniques that enhance the performance, reliability, and user experience of both the explanation and question-answering workflows. These techniques ensure the system not only generates accurate and simplified outputs but also handles real-world documents and user behavior robustly.

---

1. Section-Aware Explanation with Prompt Engineering

Rather than summarizing an entire paper in one block, SciBot isolates and explains specific sections (e.g., Abstract, Methods, Results). This:

- Increases explanation accuracy
- Allows users to focus on parts of interest
- Reduces token usage for large models

Each section is processed through a **structured prompt** that ensures output is returned as a clean JSON object. This format facilitates easier post-processing and display on the frontend.

---

2. Structured Output Parsing

To guarantee clean and interpretable model responses, SciBot uses LangChain's `StructuredOutputParser`. This ensures:

- Outputs follow a strict schema (e.g., `{ "explanation": "..." }`)
- Error handling can detect and recover from malformed outputs
- Responses can be directly mapped to front-end sections without additional parsing

---

3. Conversational Memory with Context Preservation

The chatbot uses `ConversationBufferMemory` from LangChain, allowing it to:

- Track previous questions and responses
- Support multi-turn conversations
- Provide more human-like dialogue by recalling context from earlier turns

This is especially useful when users ask follow-up questions or refer to previously discussed sections of the paper.

4. Semantic Chunking for Long Documents

The RAG pipeline applies overlapping **semantic chunking** to long documents. This ensures:

- Relevant information is preserved even across chunk boundaries
- Retrieval remains accurate even for questions that span multiple ideas
- Gemini receives the most contextually relevant content

---

5. Graceful Fallbacks and Robust Error Handling

Multiple layers of exception handling have been implemented across modules:

- Text extraction handles empty or malformed PDFs
- Model output validation ensures only well-structured content is shown to users
- Explanation modules include fallback heuristics when JSON parsing fails

---

6. Lightweight and Efficient Frontend Integration

While not technically an “ML technique,” integrating explanation and chat features with a **pure HTML, CSS, and JavaScript frontend** gives SciBot:

- Fast loading times

- Broad compatibility (no complex frontend framework dependencies)
- Flexibility for embedding into other platforms (e.g., LMS, documentation portals)

MLOps and Deployment

SciBot is built with modularity and scalability in mind, using a lightweight yet powerful stack that supports fast iteration, reliable deployment, and future extensibility.

1. Backend: FastAPI

- **Framework:** FastAPI (Python)
- **Role:** Exposes RESTful endpoints for explanation and chat functionality
- **Features:**
  - High-performance async request handling
  - Easy OpenAPI (Swagger) docs auto-generation
  - Lightweight and suitable for both local and cloud deployment
  - Modular route structure for different services (e.g., `/explain`, `/chat`, `/upload`)

2. Frontend: HTML, CSS, JavaScript

- **Purpose:** Simple and clean user interface for interaction
- **Key Features:**
  - File upload UI for submitting PDF papers
  - Toggle between section explanations and live chat interface
  - Dynamically rendered outputs (highlighted sections, chatbot responses)
  - No heavy frontend framework required — lightweight, portable, and embeddable.

3. Deployment & Scalability

While deployment specifics depend on your target environment, SciBot can be hosted in the following setups:

- **Local Hosting:** For development or academic demonstration
- **Cloud Deployment:** Easily containerized using Docker for deployment to platforms like:
  - **Render**
  - **Heroku**
  - **Google Cloud Run**
  - **AWS EC2 or Lambda**

4. API Key and Environment Management

- Gemini API keys are passed securely to the backend using environment variables or `.env` files.
- Model configurations (e.g., temperature, max tokens, number of retrieved chunks) are parameterized for easy tuning and experimentation.

Conclusion

SciBot represents a powerful and practical application of modern AI capabilities to bridge the gap between complex academic literature and accessible understanding. By combining structured section-wise explanation, conversational question answering, and session-based memory, the system enables users—from students to researchers—to engage meaningfully with scientific papers in a way that is both intuitive and efficient.

Leveraging advanced language models like **Gemini 1.5 Flash** and **Mistral**, semantic embeddings with **MiniLM**, and scalable infrastructure via **FastAPI**, SciBot delivers high-quality, context-aware responses with a streamlined user experience. The modular design ensures flexibility, maintainability, and future extensibility.

Beyond summarization, SciBot encourages interactive learning and critical inquiry by allowing users to ask tailored questions and receive grounded, accurate answers—all while preserving the logical flow of the original paper. Whether for academic review, learning support, or interdisciplinary exploration, SciBot offers a valuable AI-driven companion for navigating the growing landscape of scientific knowledge.

As a next step, future enhancements could include support for multi-document analysis, user personalization, citation tracing, and deeper integration with academic databases—further pushing the boundaries of how we interact with scientific content in the age of AI.