KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

AN AUTONOMOUS INSTITUTION - ACCREDITED BY NAAC WITH 'A' GRADE

Narayanaguda, Hyderabad.

Project Team No.: 33| 4001 |Research Paper Summarization

Team Members

1. Akshaya Batharaju : 23BD1A6646
2. Garige Yashasree : 23BD1A664P
3. Varshini Gopaldas :23BD1A664Q
4. Sampangi Vaishnavi :24BD5A6612
5. Lalitha Vaishnavi :23BD1A058K

Mentors

1. Ms.Kamal Vijetha
2. Mr. Shankar

CONTENTS

Research Paper Summarization

Using RAG MODEL

INTRODUCTION

Problem Statement:

• Reading and understanding lengthy research papers is time-consuming. Students and researchers often need quick summaries and answers to specific questions.

End User Requirement:

•Quick, accurate, and easy-to-read summaries of lengthy and technical research papers.
•User must be able to choose whether he/she needs short , medium or lengthy summaries based on their time span.
•User must be able to search for the meaning of the word that he finds difficult.
•He must get accurate answers for the questions he poses.

Abstract :

This project focuses on developing an AI-based web application that generates a detailed summary of PDF documents using Large Language Model(LLM) and RAG Model. The main objective is to help users quickly understand lengthy documents without reading them completely. The project uses Python (FastAPI) for backend, Machine Learning models like google gemini pro for text summarization, and ReactJS for frontend. The solution enhances user productivity, especially for students and researchers handling large amounts of content.

Objectives:

•Reduce time spent on initial research review.
•Enhance understanding of complex topics through concise summaries.
•Offer tailored summarization options (short, medium, long).
•Simplify exploration of unknown terms via integrated tools.

Technology Stack:

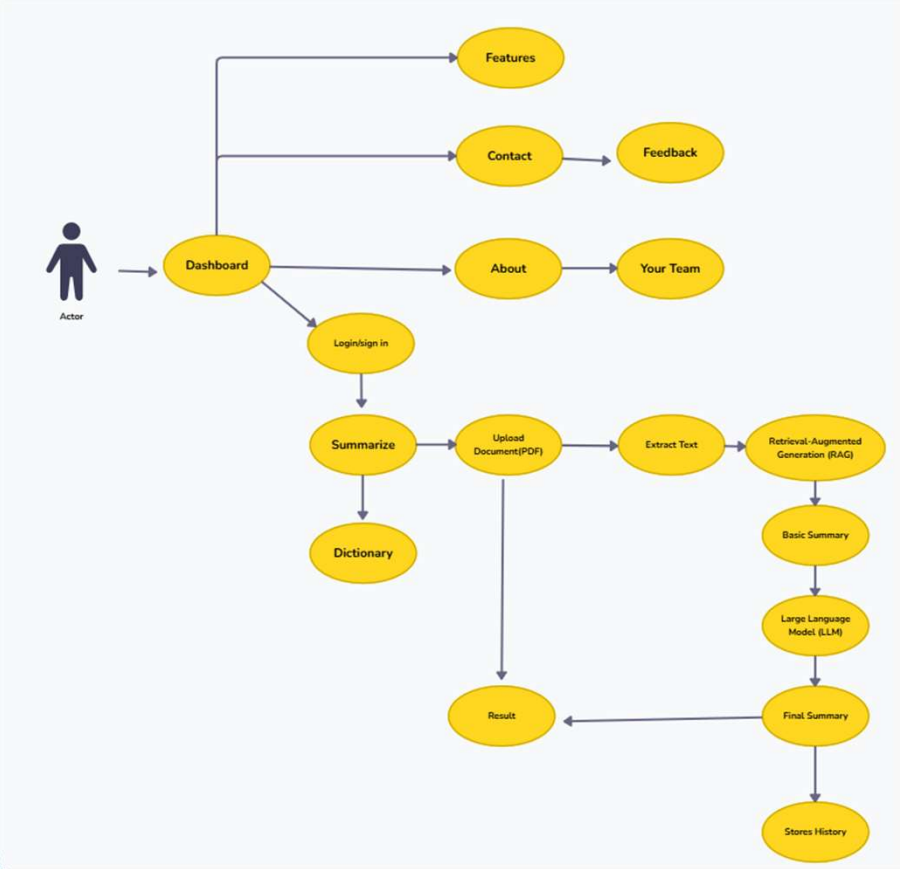
•Artificial Intelligence & Machine Learning
FIASS as Vector Database
Retrieval-Augmented Generation (RAG)
Large Language Models (LLMs) – Google Gemini Pro

REQUIREMENT

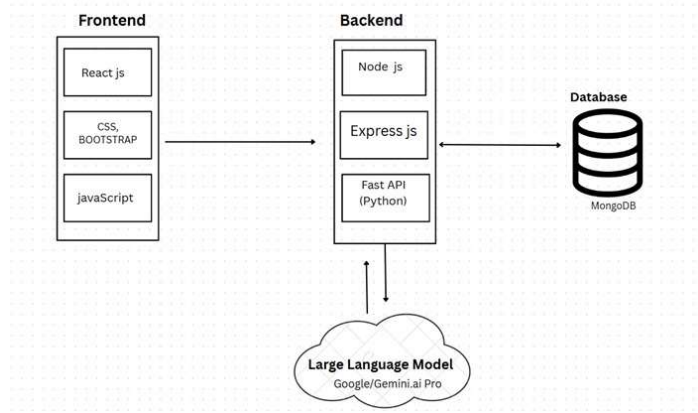
Outcome:
A web-based, intelligent summarization tool for academic content that simplifies and accelerates the research process.

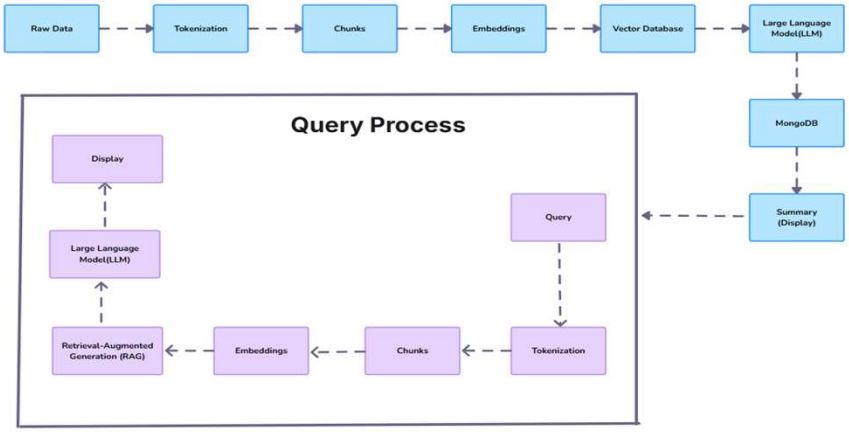
Key Features:
- **Automated summarization of uploaded research papers (PDF, DOC):**
Enables users to upload various document formats and receive instant, AI-generated summaries.
- **Adjustable summary length (short / medium / detailed):**
Allows users to choose the depth of the summary based on their needs—quick scan or deep understanding.
- **Dark and light mode themes:**
Enhances user comfort and accessibility through customizable display preferences.
- **Built-in academic dictionary :**
Offers clear definitions of technical terms to aid comprehension, especially for interdisciplinary readers.
- **User feedback submission for improvement:**
Collects input from users to refine summaries and enhance future platform updates.
- **Summarization history :**
History of the summaries is also stored..
- **AI-powered chatbot for Q&A based on uploaded content:**
Provides interactive assistance and answers user queries derived directly from the research paper content.
- **Text to Speech :**
The generated summary is converted to speech.

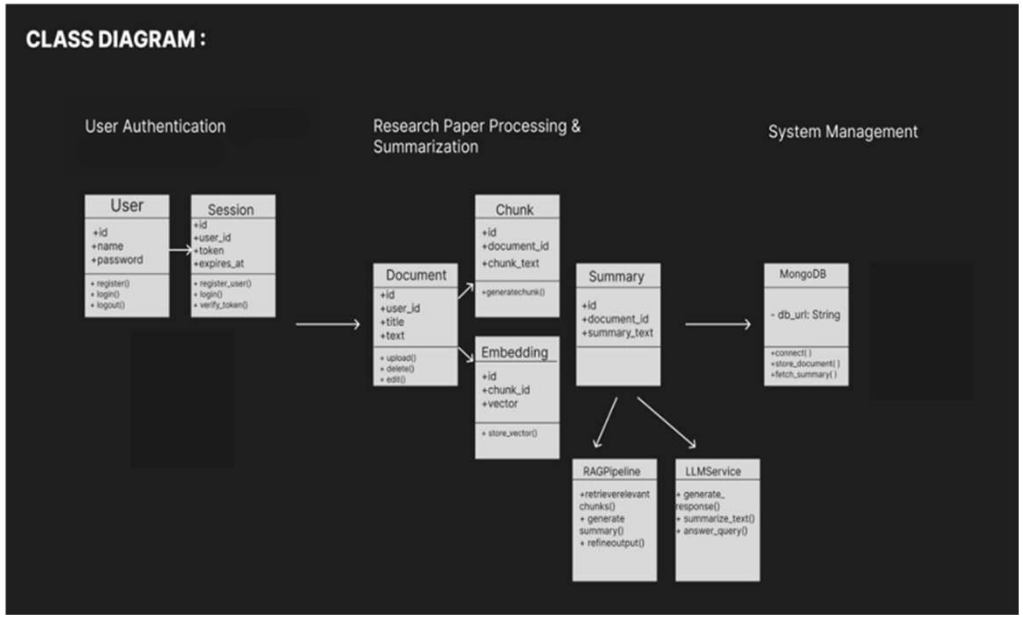
Use Case :



Architecture :

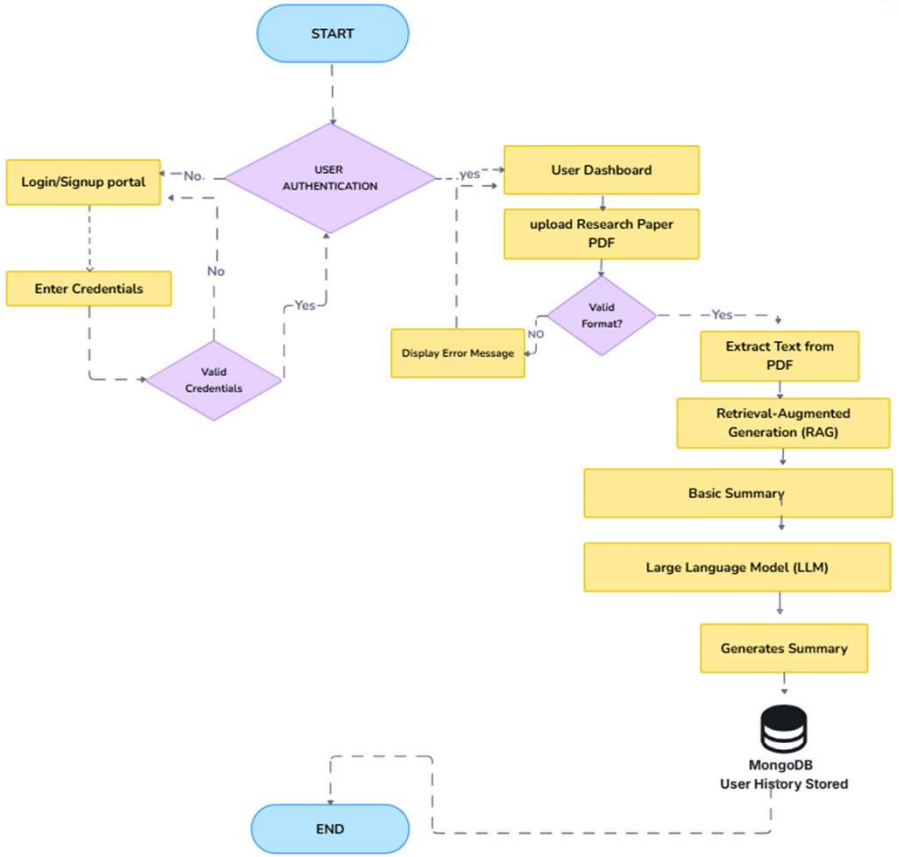




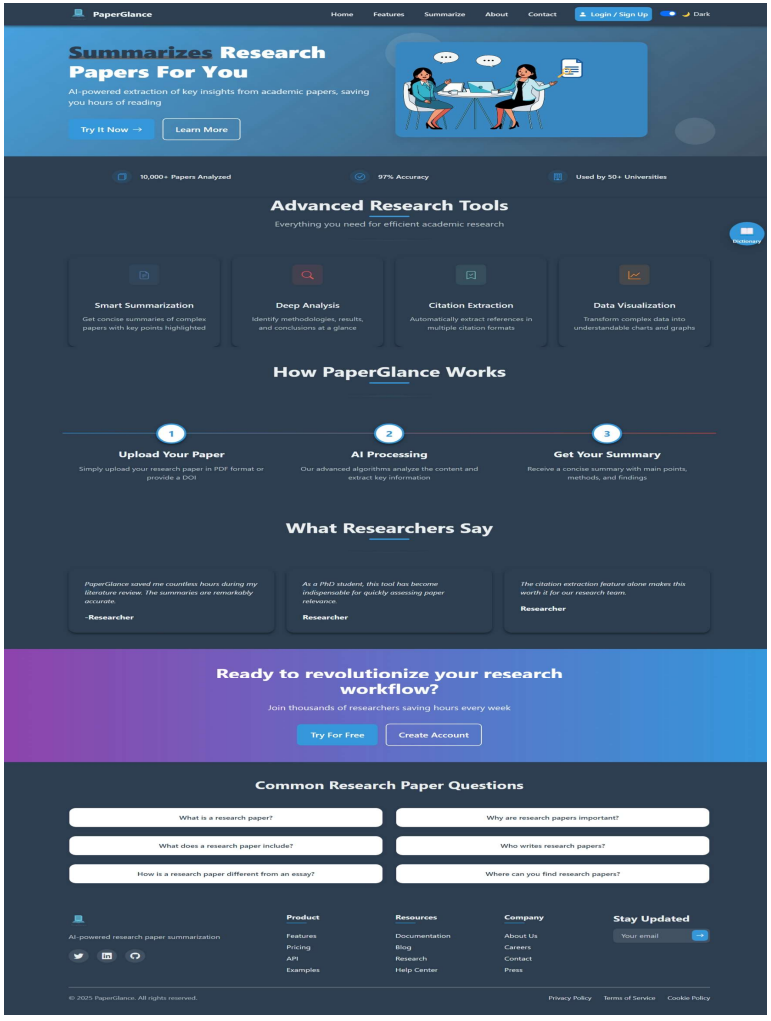


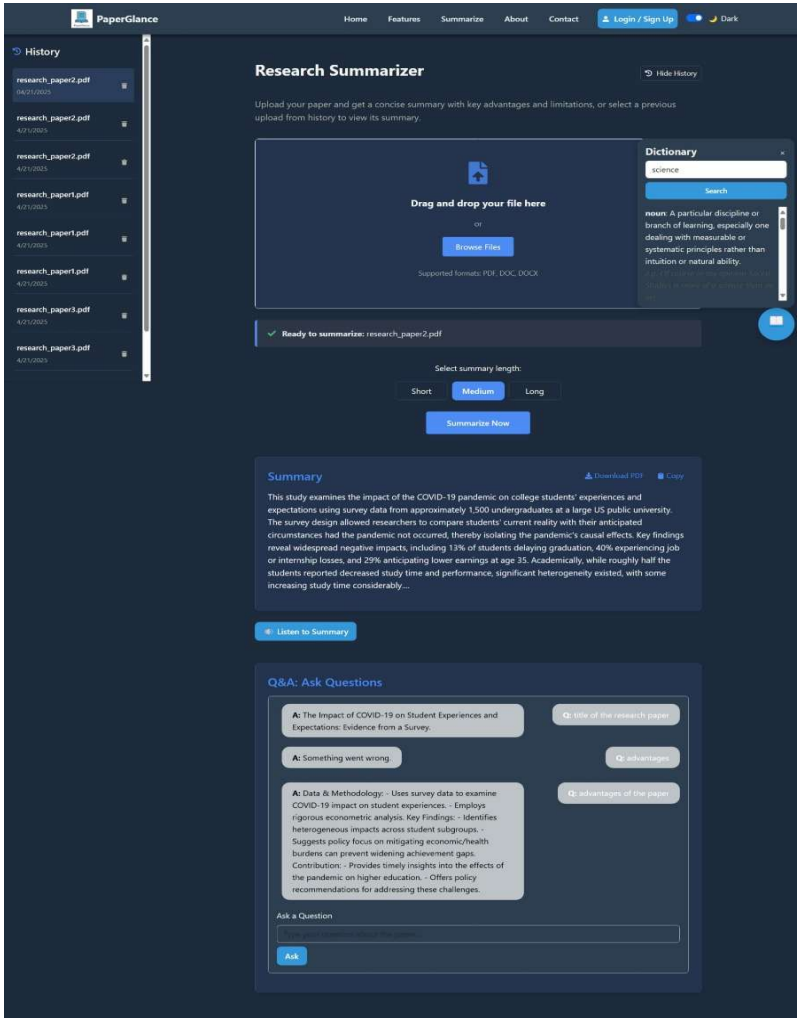
DataBase Design:

FlowChart:

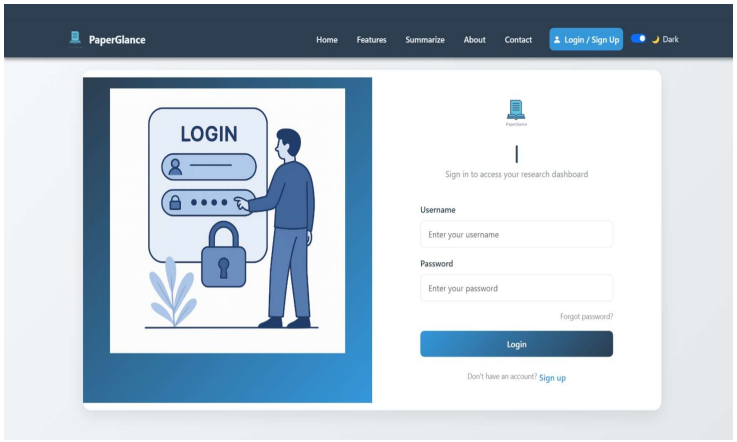


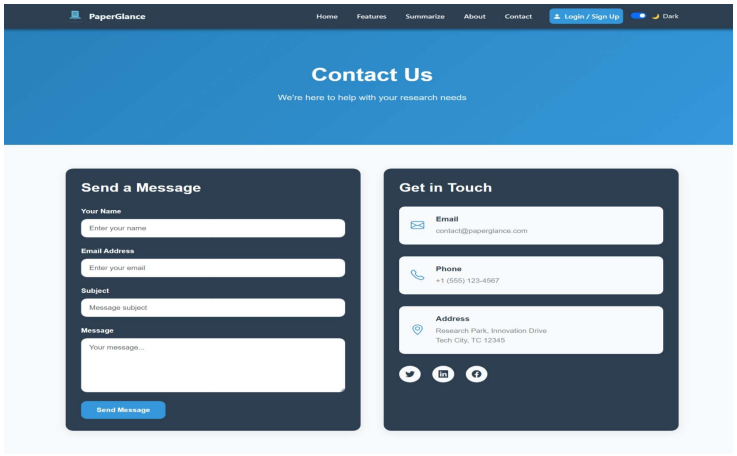
DEVELOPMENT






PaperGlance
Home
Features
Summarize
About
Contact
Login / Sign Up
Dark
About PaperGlance
Revolutionizing Academic Research Through AI
Our Mission
At PaperGlance, we're transforming how researchers interact with scholarly content. Our AI-driven platform delivers concise, accurate summaries of complex research papers, accelerating your academic workflow.
Why Choose PaperGlance
Smart Summarization
AI-powered summaries capture the essence of complex papers in minutes
Academic Integrity
Algorithms designed to maintain scholarly rigor and accuracy
Critical Analysis
Highlight key strengths and limitations for balanced evaluation
Community Driven
Developed with continuous feedback from researchers worldwide
Our Team
VG
Varshini Gopaldas
Research Specialist
LV
Laitha Vaishnavi
Research Specialist
YG
YashaSree Garige
Research Specialist
VS
Vaishnavi Sampangi
Research Specialist
AB
Akshaya Batharaju
Research Specialist
Key Features
Instant paper summarization
Citation pattern analysis
Key term identification
Research trend tracking
Transform Your Research Process
Start Free Trial


PaperGlance
Home
Features
Summarize
About
Contact
Login / Sign Up
Dark
LOGIN
Sign in to access your research dashboard
Username
Enter your username
Password
Enter your password
Forgot password?
Login
Don't have an account? Sign up


PaperGlance
Home
Features
Summarize
About
Contact
Login / Sign Up
Dark
Contact Us
We're here to help with your research needs
Send a Message
Your Name
Enter your name
Email Address
Enter your email
Subject
Message subject
Message
Your message...
Send Message
Get in Touch
Email
contact@paperglance.com
Phone
+1 (555) 123-4567
Address
Research Park, Innovation Drive
Tech City, TC 12345

Tools and Frameworks Used:

- **NLP & ML:** Google/Gemini pro 1.5
- **Frontend:** React, Bootstrap, CSS Modules
- **Backend:** FastAPI
- **Database:** MongoDB
- **Others:** Git, Postman, Google Colab, Figma (for UI prototypes)

Workflow:

Start
↓
IF Login/Signup → Enter credentials
↓
Are credentials valid?
→ No → Show error → Back to login
→ Yes
↓
User Dashboard is displayed
↓
User uploads a research paper (PDF / DOC)
↓
Is the file format valid?
→ No → Show error → Back to dashboard
→ Yes
↓
Extract text from uploaded paper
↓
Pass extracted text to RAG (Retrieval-Augmented Generation)
↓
Generate basic summary
↓
Refine using Large Language Model (LLM)
↓
Display final summary to user
↓
Store summary and user interaction in MongoDB (user history)
↓
User can log out / continue with another upload
↓
End

Frontend Pseudo code

api.js

```
SET API_URL = "http://localhost:8000"

FUNCTION uploadFile(file):
    CREATE formData with file
    TRY:
        POST to API_URL + "/upload" with formData
        RETURN response.data
    CATCH:
        LOG error
        THROW "File upload failed"

FUNCTION getSummary(query):
    TRY:
        POST to API_URL + "/summarize" with query
        RETURN response.data
    CATCH:
        LOG error
        THROW "Summary generation failed"
```

SummarizePage.js

```
FUNCTION SummarizePage():
    // State
    SET darkMode = GET_DARK_MODE_STATUS()
    SET file = NULL
    SET summaryResult = NULL
    SET error = ""
    SET historyItems = EMPTY_LIST
    SET selectedHistoryItem = NULL

    // Fetch history on mount
    WHEN COMPONENT_MOUNTS:
        SET historyItems = CALL_API("GET",
"http://localhost:8000/summaries/")

    // Handle history item click
    FUNCTION handleHistoryItemClick(item):
        SET selectedHistoryItem = item
        SET summaryResult = {summary: item.summary}
        CLEAR file

    // Handle download summary
    FUNCTION handleDownloadSummary():
        CALL_API("POST", "http://localhost:8000/download-
summary/", {summary_text: summaryResult.summary},
{responseType: 'blob'})
        TRIGGER download OF "research_summary.pdf"

    // Handle file drop/upload
    FUNCTION handleDrop(event):
        SET file = FIRST event.dataTransfer.files
        CALL handleUpload(file)

    FUNCTION handleFileChange(event):
        SET file = FIRST event.target.files
        CALL handleUpload(file)

    // Handle upload
    FUNCTION handleUpload(file):
        CALL_API("POST", "http://localhost:8000/upload/", NEW
FormData().append("file", file))
        ADD response.data TO historyItems

    // Handle summarize
    FUNCTION handleSummarize():
        CALL_API("POST", "http://localhost:8000/summarize", NEW
FormData().append("file", file))
        SET summaryResult = {summary: response.data.summary}

    // Render UI
    DISPLAY layout WITH:
        IF historyItems:
            FOR EACH item IN historyItems:
                DISPLAY item.name
                ON CLICK: CALL handleHistoryItemClick(item)
        IF NOT selectedHistoryItem:
            DISPLAY drag-and-drop area
            ON DROP: CALL handleDrop
            DISPLAY file input
            ON CHANGE: CALL handleFileChange
            DISPLAY summarize button
            ON CLICK: CALL handleSummarize
        IF summaryResult:
            DISPLAY summary
            DISPLAY download button
            ON CLICK: CALL handleDownloadSummary
            IF selectedHistoryItem:
                DISPLAY chat history
            ELSE:
                DISPLAY chat box
        IF error: DISPLAY error

    RETURN rendered component
```

Backend Pseudo code

llm_utils.py

```
SET logging TO DEBUG
SET MODEL = "gemini-1.5-pro"
SET USED_KEYS = CIRCULAR_QUEUE OF API_KEYS

FUNCTION clean_summary(summary):
    REPLACE markdown, HTML, chatty phrases WITH ""
    NORMALIZE bullet points, whitespace
    RETURN TRIM(summary)

FUNCTION make_gemini_request(prompt, api_key) WITH
RETRY (3 ATTEMPTS):
    SET url =
"https://generativelanguage.googleapis.com/v1/models/{MODE
L}:generateContent?key={api_key}"
    SET payload = {"contents": [{"parts": [{"text": "Expert
summarizer. {prompt}"}]}]}
    SEND POST request TO url WITH payload
    IF success:
        RETURN
response["candidates"][0]["content"]["parts"][0]["text"]
    ELSE:
        RAISE error

FUNCTION summarize_with_llm(chunks,
summary_length="medium"):
    IF summary_length IS "short":
        SET prompt = "Summarize in 1-2 sentences:\n" +
JOIN(chunks[:5])
    ELSE IF summary_length IS "medium":
        SET prompt = "Summarize in 2-3 paragraphs:\n" +
JOIN(chunks[:10])
    ELSE IF summary_length IS "long":
        SET prompt = "Summarize in 6-8 paragraphs with
headings:\n" + JOIN(chunks[:30])
    ELSE:
        SET prompt = "Summarize in 2-3 paragraphs:\n" +
JOIN(chunks[:5])

FOR EACH api_key IN USED_KEYS:
        TRY:
            SET raw_summary = make_gemini_request(prompt, api_key)
            RETURN format_summary(clean_summary(raw_summary),
summary_length)
        EXCEPT:
            ROTATE USED_KEYS
            IF LAST KEY: RAISE "All keys failed"

FUNCTION ask_question(question, top_chunks):
    IF question CONTAINS ["title", "author", "journal", "date",
"conference"]:
        SET prompt = "Answer in 50 words:\nQuestion:
{question}\nExcerpts:\n" + JOIN(top_chunks)
    ELSE:
        SET prompt = "Answer in 100 words with bullet
points:\nQuestion: {question}\nExcerpts:\n" + JOIN(top_chunks)

    FOR EACH api_key IN USED_KEYS:
        TRY:
            SET raw_answer = make_gemini_request(prompt, api_key)
            RETURN format_summary(clean_summary(raw_answer))
        EXCEPT:
            ROTATE USED_KEYS
            IF LAST KEY: RAISE "All keys failed"
            WAIT 1 second

FUNCTION format_summary(summary, summary_length="medium"):
    IF summary_length IN ["medium", "long"]:
        RETURN TRIM(summary)
    SET words = SPLIT(summary)
    IF LEN(words) > 100:
        SET summary = JOIN(words[:100]) + "..."
    SET sections = SPLIT(summary ON headings)
    SET formatted = ""
    FOR EACH section IN sections:
        IF MATCH "key: content":
            APPEND "{key}:\n{content}\n" TO formatted WITH bullet
points
        ELSE:
            APPEND section TO formatted
    RETURN TRIM(formatted)
```

rag_utils.py

```
# Import necessary libraries
import PDF processing library
import vector libraries (numpy, faiss)
import embedding model library
import components from llm_utils

# Constants
CHUNK_SIZE = 150  # words
CHUNK_OVERLAP = 25  # words
embedder = load SentenceTransformer model

# Document Processing Functions
function chunk_text(text):
    Split text into words
    Create overlapping chunks of CHUNK_SIZE words with CHUNK_OVERLAP
    Return list of chunks

function process_pdf(pdf_path):
    Open PDF document
    Extract full text from all pages
    Split text into chunks using chunk_text()
    Generate embeddings for all chunks
    Create FAISS vector index
    Add embeddings to index
    Return chunks and index

# Retrieval Functions
function get_top_chunks(query, chunks, index, top_k=5):
    Encode query using the embedder
    Perform vector search with FAISS index
    Return top_k most similar chunks

# RAG Implementation
function answer_with_rag(query, chunks, index, top_k=5):
    Get top_k relevant chunks using get_top_chunks()
    Log debug info

    For each API key in rotation:
        Try:
            Get answer using ask_question() from llm_utils
            Log success
            Return response
        Catch exceptions:
            Log failure
            Rotate to next API key
            If all keys failed:
                Raise exception
            Wait briefly before retrying
```

main.py

```
# Import necessary libraries
import FastAPI and related components
import MongoDB client
import PDF processing libraries
import custom modules (rag_utils, llm_utils)
import file handling and logging
# Configure application
setup FastAPI app
configure logging
setup PDF generation tool
configure CORS middleware

# Constants and Configuration
UPLOAD_DIR = "uploads"
ensure directory exists
# Setup MongoDB connection
connect to MongoDB using connection string
setup database collections for papers, summaries, and chat history

# API Endpoints
@app.post("/upload/")
function upload_pdf(file):
    Save uploaded file to UPLOAD_DIR
    Process PDF using process_pdf() from rag_utils
    Generate initial summary using summarize_with_llm()
    Create unique paper_id
    Save FAISS index to disk
    Store paper metadata in papers_collection
    Store summary in summaries_collection
    Return paper_id, filename, summary and date

@app.post("/summarize/")
function summarize_pdf(file, summary_length):
    Save uploaded file to UPLOAD_DIR
    Process PDF using process_pdf()
    Generate summary of specified length using summarize_with_llm()
    Create unique paper_id
    Save FAISS index to disk
    Store paper metadata in papers_collection
    Store summary with length info in summaries_collection
    Return paper_id and summary

@app.get("/summaries/")
function get_summaries(limit=10):
    Retrieve most recent papers from papers_collection
    Initialize empty result list and tracking set
    For each paper:
        Skip if already seen
        Get latest summary for this paper
        Format summary data with metadata
        Get chat history for this paper
        Add to result list
    Return result list

@app.post("/chat/")
function chat_with_paper(question, paper_id):
    Validate inputs
    Find paper in database
    Get or reload chunks and index
    Get top relevant chunks using get_top_chunks()
    Generate answer using ask_question()
    Store chat entry in chat_history_collection
    Return answer

@app.get("/history/{paper_id}")
function get_paper_history(paper_id):
    Validate paper_id
    Find paper in database
    Get summary for paper
    Get chat history for paper
    Format response with paper metadata, summary and chat history
    Return formatted response

@app.post("/download-summary/")
function download_summary(request):
    Parse request to get summary text
    Generate HTML content with styling
    Set PDF generation options
    Generate PDF file using pdfkit
    Return file download response

@app.delete("/delete-summary/{summary_id}")
function delete_summary(summary_id):
    Find summary in database
    Delete related chat history
    Delete summary
    Return success message

@app.post("/api/define")
function define_word(request):
    Get word from request
    Check spelling and get correction
    Query dictionary API
    Parse response for definitions
    Return formatted definitions or error message
```

PROJECT OUTCOME

Achievements

•**Developed :**Delivered a working version of the application that demonstrates core features, user flow.

•**Successfully handled summarization of research Paper across domains**

Achieved reliable and context-aware summarization for a wide range of topics including computer science, engineering, and social sciences.

• **Integrated helpful utilities** .

- Helps researchers, students, and professionals quickly understand key insights, saving time and boosting efficiency.
- Personalized User Experience :, history tracking, saved summaries, and customization options like summary length and voice mode.
- Enhanced Accessibility : Voice input and text-to-speech features improve usability for differently-abled users and those preferring audio output.
- Multilingual Document Summarization : Allows uploading papers in any language (e.g., Telugu) and generates summaries in English, enabling cross-language understanding of academic content.
- Interactive Learning Environment : AI Chat Board supports conversation-based queries, deeper insights, definitions, and explanation of paper content.
- User Feedback Integration :Feedback system helps gather user input to improve system quality and user satisfaction over time.

Key Challenges

• **Preserving technical accuracy**

Summarizing complex academic language without losing key details.

• **Speed optimization for large PDFs and documents**

Ensuring quick responses and fast loading, even with documents that have a lot of pages.

•**Designing an intuitive yet powerful UI**

Designing a simple, easy-to-use interface while including advanced features like voice-to-text from AI chat bot, and History saving for all users.

CONCLUSION

A fully functional, web-based application has been developed to summarize research papers efficiently. The platform leverages **Large Language Models (LLMs)** and **Retrieval-Augmented Generation (RAG)** techniques in the backend to generate high-quality, context-aware summaries. The frontend is built using **React**, providing a clean, responsive, and user-friendly interface. The system supports various features like adjustable summary length, dark/light mode, an integrated dictionary, chatbot assistance, and user history tracking — making it a powerful tool for students, researchers, and professionals alike.
It efficiently summarizes research papers and provides a user-friendly interface with rich academic support features.

New Skills Learned:

- Implementation of LLMs and Retrieval-Augmented Generation.
- UX design tailored for research-heavy tools.
- Working with frontend-backend integration and secure authentication.
- Managing user data securely with database systems.

Thank you!