AI RESEARCH ASSISTANT: AGENTIC PIPELINE FOR SUMMARIZATION, QUESTION ANSWERING AND VALIDATION FOR RESEARCH ARTICLES

ABSTRACT

Managing and processing as research articles, can be time-consuming and challenging, especially when summarizing and question-answering and validating information manually. Traditional approaches often lack efficiency and accuracy, making it difficult to extract meaningful insights from large volumes of data.

We present a workflow that combines Zotero for retrieving articles, a large language model (LLM) from Gemini for summarization and question-answering by using topic modelling(LDA) for generating the questions and answers use LLM and validation by using comparing the summary and original content involves the keywords and topics overlap and using Gemini embeddings to compare the similarity by using cosine to visualize the heatmap, and LangChain agents for coordinating tasks. The system automates text extraction from PDFs, generates summaries, answers questions , and validates outputs for accuracy and relevance, ensuring high-quality results with minimal manual effort.

Our approach uses Python to integrate Zotero with text extraction tools and Gemini LLM capabilities. LangChain agents manage the summarization, Question Answer, and validation steps, streamlining the entire process. This solution simplifies research articles, saves time for researchers.

Keywords: Research automation, Zotero, Gemini LLM, topic modeling (LDA), LangChain agents, text extraction, summarization, question answering, validation, cosine similarity, heatmap visualization.

INDEX

LIST OF FIGURES

1. INTRODUCTION

The exponential growth of academic literature in recent years has presented researchers with a dual challenge: accessing relevant information quickly and synthesizing it effectively. Traditional methods of manual review and summarization often prove time-consuming and inefficient, particularly when dealing with extensive collections of research papers. To address this, the AI Research Assistant, developed using Python and powered by xAI's innovative framework, offers an automated solution that integrates advanced artificial intelligence and natural language processing technologies. This tool is designed to assist researchers by fetching documents from Zotero, generating concise summaries, extracting key topics, and producing insightful questions—all while ensuring the reliability of its outputs through rigorous validation processes.

At the core of the AI Research Assistant is a Streamlit-based interface that provides an intuitive user experience, paired with a robust backend leveraging the Google Generative AI model (Gemini-1.5-Flash) and libraries such as LangChain, PyZotero, and NLTK. The system begins by retrieving PDF documents from a user's Zotero library, extracting their text, and processing it into manageable chunks for summarization. Using a custom summarization agent, it distills complex research papers into clear, concise overviews tailored for researchers seeking to grasp core contributions quickly. Beyond summarization, the tool employs topic modeling and question generation agents to identify critical themes and pose exploratory questions, enhancing the user's ability to engage deeply with the material. Validation mechanisms, including keyword overlap analysis and cosine similarity comparisons, ensure that summaries and answers remain faithful to the original content.

The significance of this tool lies in its potential to transform how researchers interact with scholarly content, saving time and fostering deeper insights. By automating repetitive tasks and providing structured outputs—such as downloadable summaries and validated Q&A pairs—the AI Research Assistant empowers users to focus on analysis and discovery rather than preliminary processing.

Moreover, the AI Research Assistant enhances collaboration and accessibility in academic research by enabling seamless integration with existing workflows. Researchers can customize the summarization depth, refine generated questions, and interact with AI-driven insights to tailor outputs to their specific needs. The system's modular architecture allows for future expansions, such as multilingual support, domain-specific fine-tuning, and integration with additional research repositories beyond Zotero. By leveraging cutting-edge AI and natural language processing techniques, this tool not only accelerates literature review but also fosters a more efficient, data-driven research environment, ultimately contributing to innovation and knowledge advancement across disciplines.

By automating the extraction, analysis, and validation of scholarly content, it reduces cognitive load and enhances productivity for researchers across various fields. The integration of machine learning techniques, such as topic modeling with Latent Dirichlet Allocation (LDA) and semantic similarity analysis using embeddings, ensures that the generated outputs maintain contextual accuracy and relevance. As artificial intelligence continues to evolve, this tool sets the foundation for more advanced research assistants capable of deeper comprehension, personalized recommendations, and adaptive learning, ultimately redefining how knowledge is processed and utilized.

1.1 EXISTING SYSTEM

Many tools already exist to help researchers summarize academic papers quickly. Scholarcy is one example—it uses AI to pull out main ideas, methods, and results from papers, turning them into short summaries or flashcards. It connects with Zotero to grab PDFs easily. Another tool, Paper Digest, summarizes open-access articles by picking out key sentences, making it faster to read long papers. These systems mostly focus on copying important parts of the text, though some are starting to rewrite summaries in simpler words.

Other tools, like Elicit and Consensus, go further by mixing summarization with research help. Elicit summarizes papers and pulls out specific details, like data or conclusions, to answer questions across multiple studies. Consensus acts like a search engine, giving short answers about what papers agree on, backed by citations. While

they don’t check if summaries are totally accurate, their use of big databases and references makes them seem reliable. Researchers like them for fast, useful insights.

Validation—making sure summaries match the original paper—isn’t a big focus yet.it checks how papers are cited (supporting or contradicting) to show if claims are trustworthy, but it doesn’t summarize. Most tools leave accuracy checking to the user or extra steps. This gap shows a need for systems that summarize, ask questions, and check everything in one go, which is what the AI Research Assistant in this report tries to do better.

This lack of built-in validation presents a critical challenge in research automation, as misinformation or misinterpretation can lead to flawed conclusions. The AI Research Assistant aims to bridge this gap by integrating summarization, question generation, and validation into a unified system. Unlike existing tools, it employs advanced AI techniques such as **cosine similarity, keyword overlap analysis, and embedding-based validation** to cross-check summaries and answers against the original text. This ensures that extracted insights are not only concise but also **accurate, reliable, and contextually appropriate**. Furthermore, its ability to generate topic-based questions enhances comprehension, allowing researchers to engage more deeply with the material rather than passively consuming summaries.

By combining **automated summarization, topic modeling, Q&A generation, and rigorous validation**, the AI Research Assistant represents a more holistic approach to research synthesis. Future improvements could include **multilingual support, domain-specific fine-tuning, and integration with other academic databases**, further expanding its usefulness across disciplines. As research continues to evolve, tools that go beyond simple summarization—offering contextual awareness, verification, and interactive exploration—will play an increasingly important role in academic workflows.

1.2 PROPOSED SYSTEM

The proposed system, called the AI Research Assistant, is a tool designed to help researchers handle academic papers more easily. It connects to Zotero to fetch PDFs, summarizes them into short, clear overviews, and creates questions to explore the content further. Unlike other tools, it also checks if the summaries and answers stay true to the original paper, making sure the information is accurate. Built with Python and Streamlit, it's simple to use and runs on AI power from Google's Gemini-1.5-Flash model.

Here's how it works: First, it grabs a PDF from Zotero, pulls out the text, and breaks it into chunks. A summarization agent reads each chunk and writes a summary in its own words. Then, a second agent finds key topics and makes three thoughtful questions. Users can get answers based on the summary or the original text, and a validation agent compares everything—checking keywords, topics, and answer similarity—to make sure nothing important is lost. All this happens in a clean interface where users pick options and see results fast.

This system improves on existing tools by doing more in one place. While tools like Articles summarize well and answers questions, they don't check accuracy automatically. The AI Research Assistant combines summarization, question generation, and validation, saving time and boosting trust in the results. It's meant to help researchers focus on understanding papers, not just reading them, making it a step up from what's out there now.

The **AI Research Assistant** is an advanced tool designed to streamline the research process by automating text extraction, summarization, question generation, and validation. It integrates with **Zotero** to retrieve academic papers, summarizes key findings, and ensures that generated insights remain accurate and reliable. Unlike existing solutions, it uniquely combines summarization, question-answering, and validation into a single workflow, reducing manual effort for researchers.

2.LITERATURE SURVEY

The rapid growth of scientific literature has overwhelmed traditional manual review processes, prompting the development of AI-powered tools to assist researchers in analyzing, summarizing, and synthesizing knowledge. This literature survey examines recent advancements in AI-driven research assistance, focusing on tools that automate literature review, summarization, and question generation. The survey aims to contextualize [your project name], which leverages AI to streamline academic research workflows.

Automated summarization has emerged as a cornerstone of AI research tools. Blei et al. (2018) introduced Latent Dirichlet Allocation (LDA) for topic modeling, enabling concise extraction of key themes from large corpora, though it struggles with nuanced context. More recently, Lewis et al. (2020) developed BART, a transformer-based model for abstractive summarization, achieving high ROUGE scores on news datasets but requiring adaptation for academic texts. In healthcare, Dernoncourt et al. (2017) applied RNNs to summarize clinical papers, reporting a 30% reduction in reading time for practitioners. These studies underscore the importance of domain-specific training data, a challenge [your project] addresses through integration with Zotero libraries.

AI-driven question generation enhances researcher interaction with literature. Du et al. (2017) proposed a sequence-to-sequence model to generate questions from text, improving comprehension in educational settings with a BLEU score of 0.42. Wang et al. (2021) extended this to scientific domains using BERT, achieving higher relevance in question formulation but noting limitations in capturing interdisciplinary insights. Heilman & Smith (2010) pioneered rule-based approaches for question generation, offering interpretability at the cost of scalability. These efforts align with [your project's] use of topic-extracted questions to probe deeper into research content, though scalability across diverse fields remains underexplored.Ensuring the reliability of AI outputs is critical for research applications. Zhang et al. (2022) employed cosine similarity on embeddings to validate summary accuracy against original texts, reporting a 0.85 similarity threshold for trustworthiness. Similarly, Li et al. (2023) used TF-IDF and LDA to assess keyword and topic overlap, identifying a 50% overlap as a validity benchmark. Validation in question-answering systems.

3. SYSTEM ANALYSIS

3.1 FUNCTIONAL REQUIREMENTS

1. **Integration with Zotero Library**

- The system shall connect to a Zotero account using a user-provided API key and library ID.
- The system shall retrieve a list of available research papers and their metadata (e.g., titles, keys) from the Zotero library.

2. **PDF Retrieval and Processing**

- The system shall download PDF attachments from selected Zotero items.
- The system shall extract text content from downloaded PDFs using a PDF parsing library (e.g., PyMuPDF).

3. **Text Summarization**

- The system shall summarize extracted text from PDFs into concise, structured summaries.
- The system shall allow users to process large documents by chunking text (e.g., 10,000 characters per chunk) and summarizing each chunk individually.
- The system shall provide a downloadable summary file in text format.

4. **Topic Extraction**

- The system shall identify key topics from the original text using topic modeling techniques (e.g., LDA).
- The system shall present extracted topics to the user for review.

5. **Question Generation**

- The system shall generate a set of three thoughtful, research-oriented questions based on extracted topics.
- The system shall display generated questions in a numbered list format for user interaction.

6. **Question Answering**

- The system shall provide answers to generated questions using both summarized content and original text.
- The system shall differentiate between answers derived from the summary and those from the original content for comparison.

7. **Validation of Summaries**

- The system shall validate summaries by calculating keyword overlap and topic overlap with the original text.
- The system shall provide validation metrics (e.g., percentage overlap) and a validity status (e.g., valid if overlap > 50%).

8. **Validation of Answers**

- The system shall compare answers generated from summarized content against those from original content using embedding-based similarity (e.g., cosine similarity).
- The system shall display a similarity score and a heatmap visualizing answer consistency.

9. **User Interface**

- The system shall provide a web-based interface (e.g., via Streamlit) with navigation options for summarization, question generation, and validation.
- The system shall include a progress bar for long-running tasks (e.g., summarization of multiple chunks).
- The system shall allow users to select one or multiple PDFs from Zotero for processing.

10. **Error Handling and Feedback**

- The system shall display error messages for failed operations (e.g., PDF download issues, summarization errors).

- The system shall implement retry logic (e.g., up to 5 retries) for API calls to handle transient failures.

3.2 NON-FUNCTIONAL REQUIREMENTS

1. **Performance:**
    - Process and summarize a 10-page PDF in under 5 minutes**.**
    - Handle PDFs up to 20 pages without crashing**,** even on minimum hardware (4-core CPU, 16 GB RAM)**.**
2. **Usability:**
    - Provide a clear, user-friendly Streamlit interface**.**
    - Ensure compatibility with modern browsers**.**
    - Display progress bars and error messages for smooth user experience.
3. **Reliability:**
    - Generate accurate summaries and answers at least 80% of the time (based on validation checks).
    - Maintain stable connections to Zotero and Google AI, assuming a reliable internet connection.
4. **Security:**
    - Keep API keys and user data secure, ensuring they are not displayed on-screen**.**
5. **Compatibility:**
    - Run on Windows, macOS, and Linux without requiring complex setup beyond installing Python libraries.

3.3 HARDWARE REQUIREMENTS

1. **CPU (Intel Core i5 - 8th Gen)**

   The Intel Core i5 (8th Gen) processor is a mid-range CPU capable of handling moderate computational tasks.It is sufficient for running **Python-based NLP applications**, text processing, and AI inference tasks using **Gemini API** .

2. **RAM (8GB)**

The 8GB RAM can support document processing, embeddings, and query execution without significant slowdowns. However, large PDFs or extensive datasets may require optimization (e.g., chunking text effectively to avoid memory overload). If working with multiple large documents simultaneously, upgrading to 16GB RAM would improve performance.

3. **Storage (256GB SSD)**

As long as the dataset remains moderate, storage should not be a bottleneck.For large-scale projects, external storage or cloud-based solutions (e.g., Google Drive, AWS, or database integration) can help.

4. **GPU (Integrated Graphics)**

Since the project primarily uses text-based processing, GPU acceleration is not required.If large-scale embeddings or AI inference were needed, a dedicated GPU (NVIDIA) would be beneficial.For now, the system's integrated GPU is sufficient for this project.

5. **Internet Connection**

A stable internet connection is crucial for API-based tasks, including using Google Gemini API for text generation.

Latency-sensitive operations, such as real-time document retrieval, benefit from a reliable network.

3.4 SOFTWARE REQUIREMENTS

1. **Programming Language:**
    - Python **3.9 or higher**.
2. **Key Libraries & Dependencies:**
    - **Streamlit** – Web interface.
    - **PyZotero** – Connects with Zotero's API.
    - **LangChain** – Manages AI agents and tools.
    - **PyMuPDF (fitz)** – Reads and extracts text from PDFs.
    - **NLTK & Gensim** – Text processing and topic modeling (LDA).
    - **Scikit-learn & NumPy** – Mathematical computations for validation.
    - **Pandas** – Data handling and processing.
    - **Matplotlib & Seaborn** – Data visualization and heatmaps.
    - **Requests** – Handles web API calls.
3. **AI Model & API Integration:**
    - **Google Generative AI SDK (langchain-google-genai)**.
    - **API key for Gemini-1.5-Flash model** (required for AI-powered tasks).
4. **Operating System Compatibility:**
    - Runs on **Windows, macOS, and Linux**.
5. **Internet Requirements:**
    - Active **internet connection** required to fetch PDFs from Zotero and use Google AI services.
6. **Installation & Setup:**
    - All required libraries can be installed via **pip**.
    - **Zotero setup required** with a **user account and API key** to link the library.

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

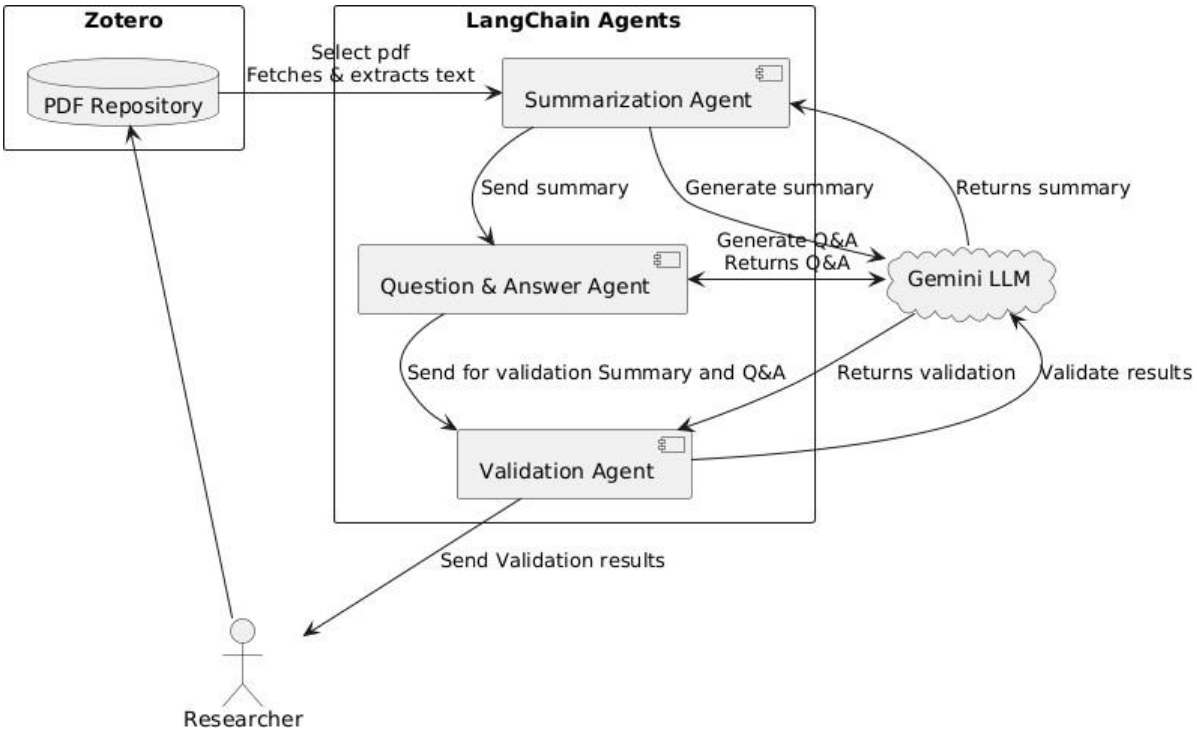


FIG 4.1: SYSTEM ARCHITECTURE

The diagram illustrates the **workflow of a research assistant system** that integrates **Zotero, LangChain agents, and the Gemini LLM** to automate summarization, question-answering, and validation of research papers.

1. Zotero - PDF Repository

- The **Zotero PDF Repository** serves as the primary storage for research papers.
- The researcher selects a PDF from Zotero, and the system fetches and extracts the text for further processing.

2. LangChain Agents

The system consists of **three main agents** that work together to process, analyze, and validate the extracted text:

(a) Summarization Agent

- Fetches and extracts text from the selected PDF.
- Sends the extracted text to **Gemini LLM** to generate a summary.
- Receives and stores the summary for further use.

(b) Question & Answer Agent

- Uses the summary to generate topic-based **questions and answers (Q&A)** with the help of **Gemini LLM**.
- Receives the generated Q&A and stores them.
- Sends the summary and Q&A to the Validation Agent for accuracy checking.

(c) Validation Agent

- Compares the **original text, summary, and Q&A** to ensure accuracy and relevance.
- Uses **Gemini LLM** to validate the generated content.
- Returns validated results to the system.

3. Gemini LLM - AI Processing Unit

- The **Gemini LLM** plays a crucial role in three areas:
    1. Generating summaries based on extracted text.
    2. Generating **questions and answers** based on topic modeling.
    3. Validating the accuracy and relevance of the generated summary and Q&A.

4. Researcher Interaction

- The **researcher** initiates the process by selecting a PDF from **Zotero**.
- After the system completes **summarization, Q&A generation, and validation**, the researcher receives the final results.

5. UML DIAGRAMS

5.1 USECASE DIAGRAM

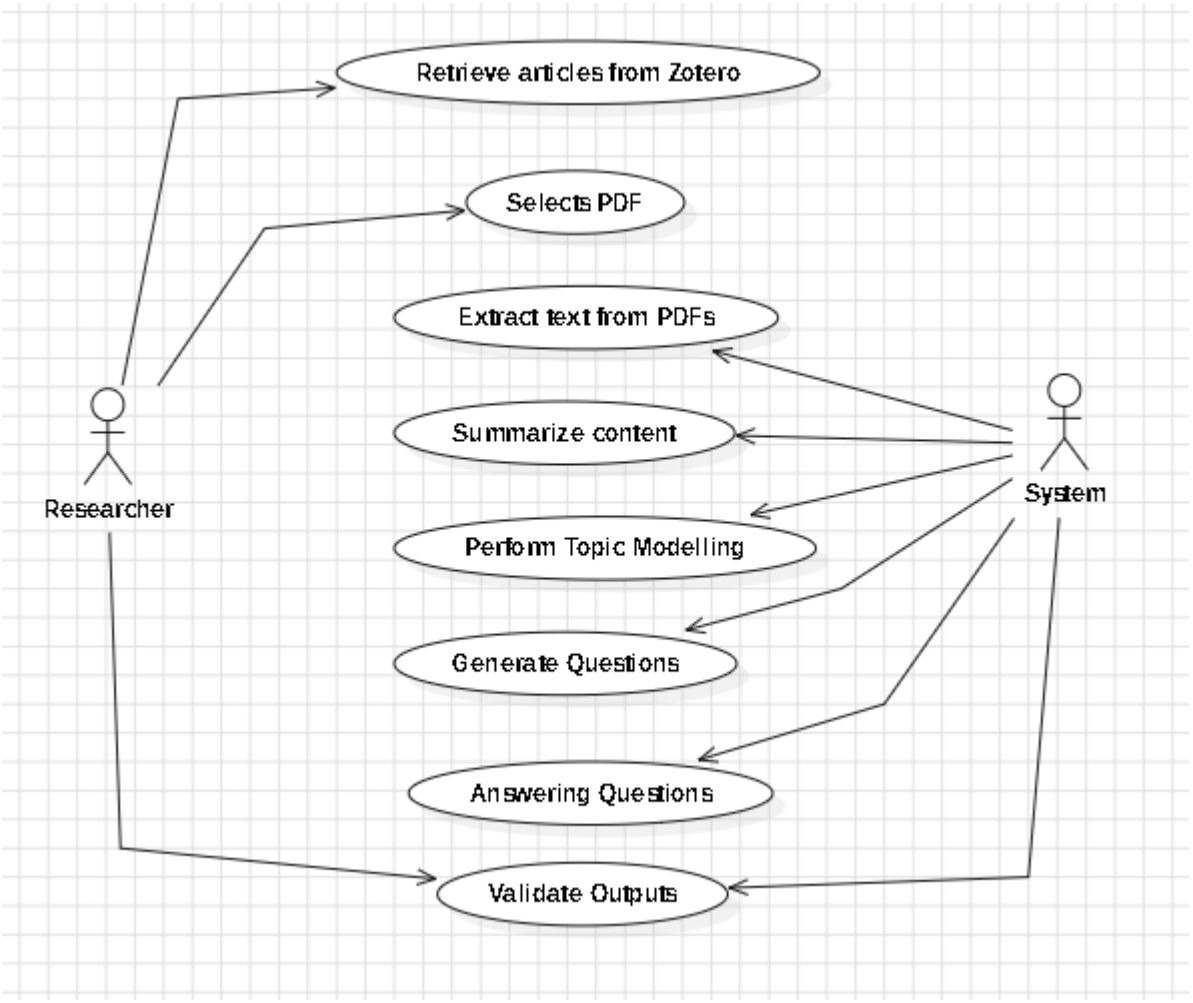


FIG 5.1: Use Case Diagram

Actors:

1. **Researcher (User):** The individual interacting with the system to process research papers.
2. **System:** The AI-powered system that automates text extraction, summarization, topic modeling, Q&A generation, and validation.

Use Cases :

1. **Retrieve Articles from Zotero**
   - The **researcher** interacts with **Zotero**, a reference management tool, to fetch relevant research articles.
   - Zotero's API retrieves bibliographic metadata and full-text PDFs from the researcher's personal library or shared repositories.
   - This step ensures that only the most relevant academic sources are selected for analysis.
2. **Select PDF**
   - The researcher selects a specific **PDF document** from the retrieved Zotero articles.
   - The selection process allows the researcher to focus on papers that align with their area of interest.
3. **Extract Text from PDFs**
   - The **system extracts raw text** from the selected PDF using a **PDF parsing library** (e.g., PyMuPDF).
   - This extracted text serves as the foundational content for further processing, including summarization and topic analysis.
4. **Summarize Content**
   - A **summarization agent**, powered by an **LLM (Gemini API or another LLM model)**, generates a **concise summary** of the extracted text.
   - Summarization techniques such as **MapReduce or refine-based summarization** are used to create high-quality, structured outputs.
   - The summary helps researchers quickly grasp the **core findings and contributions** of the research paper.
5. **Perform Topic Modeling**
   - The system applies **Latent Dirichlet Allocation (LDA)** using **NLTK and Gensim** to identify the key topics present in the document.
   - Topic modeling helps in **structuring unstructured text** by revealing recurring themes and subject areas.
   - This step is crucial for automatically identifying the **main areas of discussion** within the research paper.

6. **Generate Questions**

- Based on the discovered **topics**, the system generates **contextually relevant questions** to highlight critical aspects of the research paper.
- These questions serve multiple purposes:
  - **Enhancing comprehension** by focusing on key points.
  - **Facilitating knowledge extraction** for researchers looking to explore specific concepts.
  - **Preparing educational materials** such as quizzes or discussion prompts.

7. **Answering Questions**

- The system processes the generated questions and provides **AI-driven answers** using an **LLM-powered Q&A agent** (Gemini LLM).
- The Q&A model retrieves **accurate and context-aware responses** based on the extracted content from the paper.

8. **Validate Outputs**

- A **validation agent** is used to assess the **accuracy, consistency, and reliability** of the generated outputs.
- The agent compares:
  - **Summarized content vs. Original text** → Ensures that no critical information is lost.
  - **Generated Q&A vs. Original text** → Verifies that the answers align with the source material.
- The validation process utilizes **embedding-based similarity measures (e.g., using gemini)** to ensure **semantic accuracy**.

5.2 CLASS DIAGRAM

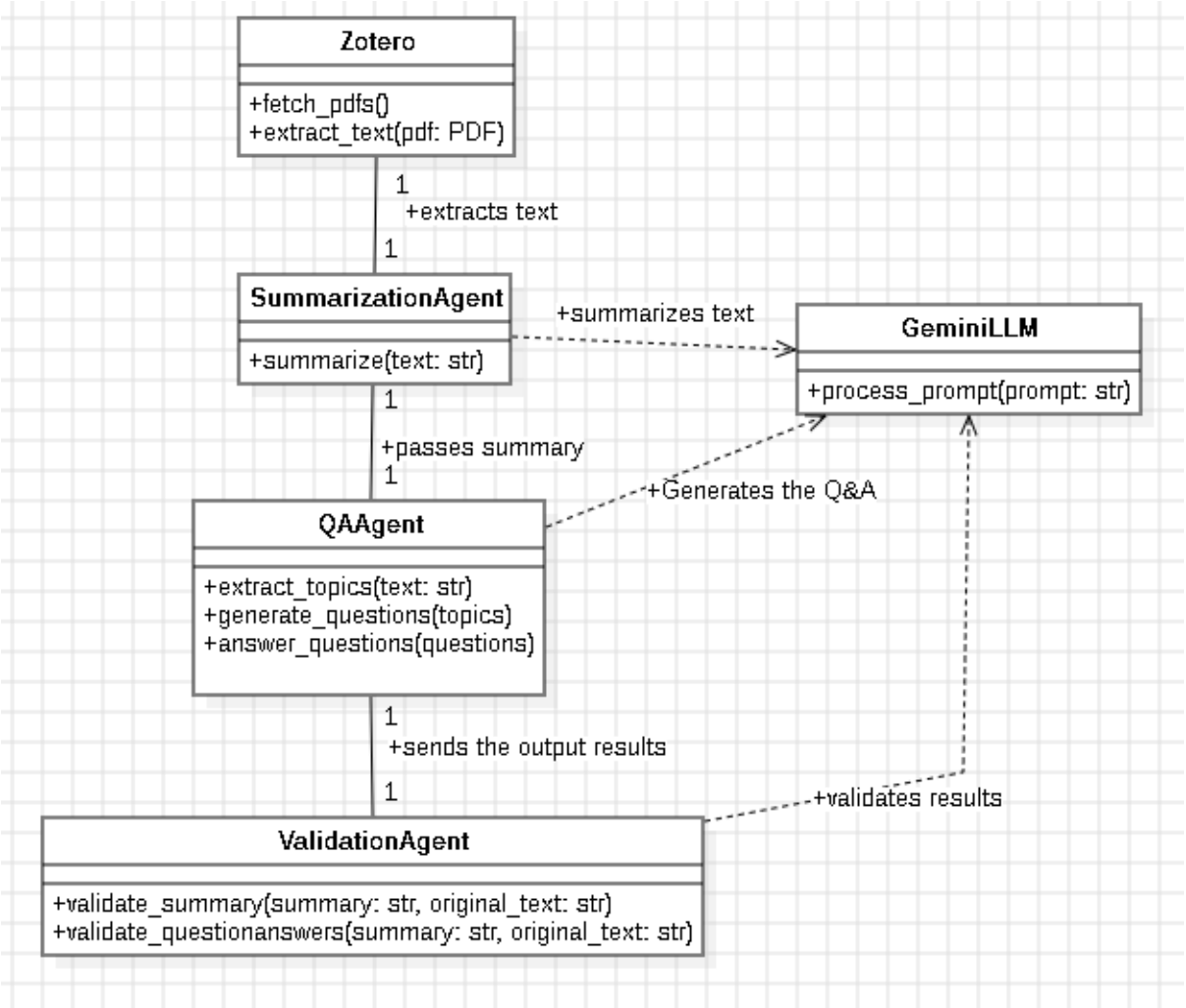


FIG 5.2 Class Diagram

This **class diagram** represents the architecture of a **LangChain-based research automation system**, detailing its main components, their attributes, and interactions.

1. Zotero

- **Functions:**
  - fetch_pdfs(): Retrieves PDFs from the Zotero reference library.
  - extract_text(pdf: PDF): Extracts text from the selected PDF.
- **Interaction:**
  - Connected to the **SummarizationAgent** for processing extracted text.

2. Summarization Agent

- **Functions:**
    - summarize(text: str): Generates a concise summary from the extracted text.
- **Interaction:**
    - Receives extracted text from **Zotero**.
    - Sends the summary to **GeminiLLM** for processing.
    - Passes the summary to **QAAgent** for question generation.

3. GeminiLLM

- **Functions:**
    - process_prompt(prompt: str): Processes text-based prompts for summarization and Q&A generation.
- **Interaction:**
    - Assists the **SummarizationAgent** in summarizing text.
    - Assists the **QAAgent** in generating **questions and answers**.
    - Assists the **ValidationAgent** in verifying results.

4. QAAgent (Question & Answer Agent)

- **Functions:**
    - extract_topics(text: str): Identifies key topics from the text using **topic modeling (LDA, NLTK, Gensim)**.
    - generate_questions(topics): Generates questions based on identified topics.
    - answer_questions(questions): Uses the **Gemini LLM** to provide answers.
- **Interaction:**
    - Receives summarized content from **SummarizationAgent**.
    - Uses **GeminiLLM** to generate and answer questions.
    - Sends Q&A results to the **ValidationAgent** for verification.

5. ValidationAgent

- **Functions:**

    - validate_summary(summary: str, original_text: str): Compares the generated summary with the original text to ensure accuracy.
    - validate_questionanswers(summary: str, original_text: str): Evaluates the relevance of generated **Q&A pairs** by comparing them with the original text.
- **Interaction:**
    - Receives summary and Q&A results from **QAAgent**.
    - Uses **GeminiLLM** to verify output quality.
    - Provides final validation feedback.

Key Relationships

- **Zotero → Summarization Agent**: Zotero extracts text, which is passed to the summarization agent.
- **Summarization Agent → Gemini LLM**: Uses Gemini LLM for text summarization.
- **Summarization Agent → QA Agent**: The summary is passed to the QA Agent.
- **QA Agent → Gemini LLM**: Generates Q&A using Gemini.
- **QA Agent → Validation Agent**: Sends Q&A results for validation.
- **Validation Agent → Gemini LLM**: Uses Gemini for validation checks.

5.3 SEQUENCE DIAGRAM

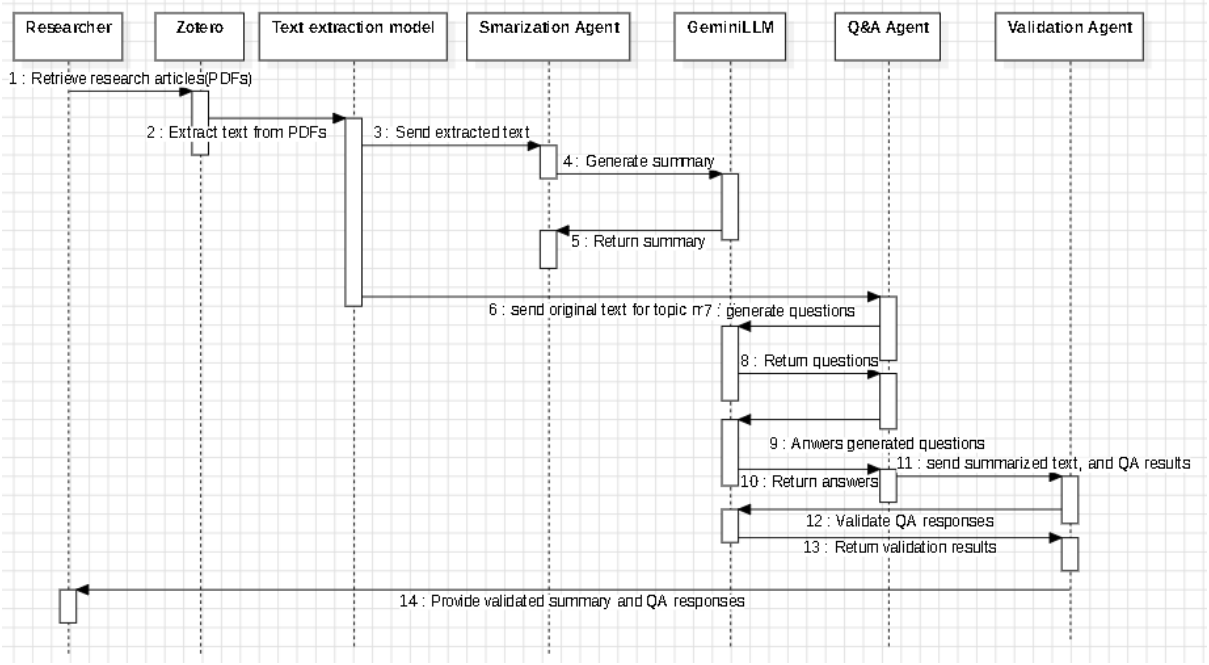


FIG 5.3: Sequence Diagram

This diagram represents the workflow of your LangChain-based research automation system using multiple agents. Here's a simple breakdown:

1. **Researcher Initiates Process**
   - The **researcher selects** relevant **research articles (PDFs) from Zotero**, a reference management tool.
   - Zotero allows researchers to efficiently **organize and manage their literature**, ensuring they have access to high-quality academic sources.
   - Once the **desired PDFs are selected**, the system begins the **automated processing pipeline**.

2. **Text Extraction**
   - **Zotero retrieves and extracts text** from the selected PDFs.
   - The extracted text is **preprocessed** (removing metadata, references, and non-relevant content) before analysis.
   - The **text extraction model** ensures that clean, structured text is available for further **summarization, topic modeling, and Q&A generation**.

- If necessary, **OCR (Optical Character Recognition) techniques** can be applied for **scanned PDFs** to ensure complete text retrieval.

3. **Summarization Process**
   - The **extracted text** is **sent to the Summarization Agent**, responsible for generating a concise and meaningful summary.
   - The **Summarization Agent interacts with the Gemini LLM**, sending a summarization prompt to the model.
   - The **LLM processes the text** and returns a well-structured **summary** that captures the **key points, findings, and contributions** of the research paper.
   - The **system supports different summarization strategies**, such as:
     Extractive summarization (selecting the most important sentences).
     Abstractive summarization (paraphrasing content in a more readable format).
   - The generated summary allows researchers to **quickly grasp the core insights** without having to read the entire paper.

4. **Question Generation and Answering**
   - The **original text** is used as input for **topic-based question generation** through the **Q&A Agent**.
   - **Steps in the process:**
     - **Topic Extraction:** The Q&A Agent applies **LDA-based topic modeling (NLTK/Gensim)** to discover the key themes of the research paper.
     - **Question Generation:** The **Gemini LLM** generates **contextually relevant questions** based on the identified topics.
     - **Answering Questions:** The **Q&A Agent** uses the **Gemini LLM** to generate **accurate and well-formed answers** for the questions.
   - This step enhances **knowledge comprehension** and allows for deeper exploration of key concepts within the research paper.
   - The **generated Q&A pairs** can be used for **educational purposes, literature reviews, or research discussions**.

5. **Validation**
    - The **summarized text and generated Q&A responses** are **sent to the Validation Agent** for quality assessment.
    - The Validation Agent performs **accuracy and relevance verification** using:
        - **Semantic similarity analysis** (using **gemini embeddings** to compare original text vs. summarized text).
        - **Content consistency checks** (ensuring summaries and answers align with the original text).
        - **Fact verification** (detecting potential hallucinations or misinformation in AI-generated content).
    - The agent **flags inconsistencies or missing details**, ensuring that only **high-quality and reliable outputs** are delivered to the researcher.
6. **Final Output**
    - After successful validation, the system **delivers the final output** to the **researcher**, including:
        - **A refined and validated summary** of the research paper.
        - **A set of topic-based questions with accurate AI-generated answers**.
        - **Validation results indicating confidence scores and quality metrics**.
    - Researchers can now **review, interpret, and utilize** these outputs in their academic work, enabling **efficient literature analysis and knowledge extraction**.

This research automation workflow streamlines the process of analyzing academic papers by integrating **Zotero, Gemini LLM, and AI-driven agents**. The researcher begins by selecting research articles from Zotero, and the system extracts text from the PDFs. The **Summarization Agent** processes the extracted text using Gemini LLM to generate a concise summary. Next, the **Q&A Agent** applies topic modeling (LDA) to identify key themes, generates relevant questions, and provides AI-driven answers.To ensure accuracy, the **Validation Agent** cross-checks the summary and Q&A responses with the original text, using **semantic similarity and consistency checks**. The validated outputs, including the refined summary and Q&A pairs, are then provided to

the researcher. This automated workflow enhances research efficiency, saves time, and ensures reliable insights for academic studies.

5.4 ACTIVITY DIAGRAM

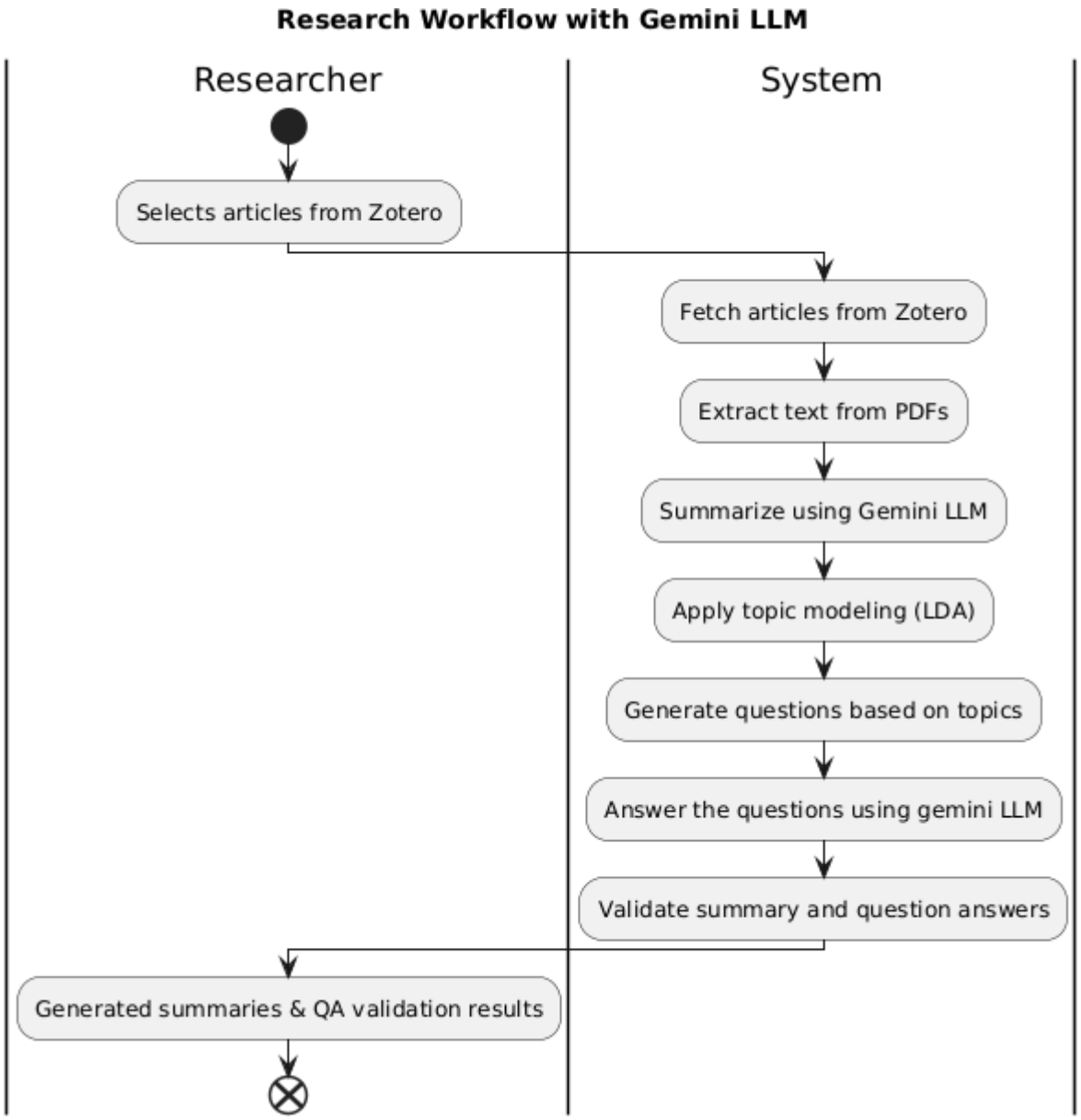


FIG 5.4: Activity Diagram

This diagram represents a streamlined research workflow using the **Gemini LLM** to automate summarization, topic modeling, and question-answering. Here's a simple breakdown:

1. Researcher's Role

- The researcher **selects articles** from **Zotero** to begin the process.

2. System's Automated Workflow

1. **Fetch Articles from Zotero**
   - Uses Zotero's API to retrieve selected research papers (PDFs).
   - Extracts metadata (title, author, etc.) for reference.
2. **Extract Text from PDFs**
   - Utilizes a PDF parsing library (e.g., PyMuPDF).
3. **Summarization using Gemini LLM**
   - Passes the extracted text to a LangChain-powered summarization agent.
   - Uses a structured prompt with a summarization technique (MapReduce, refine, etc.).
   - Outputs a concise, topic-aware summary.
4. **Topic Modeling with LDA**
   - Uses NLTK and Gensim to apply **Latent Dirichlet Allocation (LDA)**.
   - Identifies key topics and themes from the text.
   - Helps generate meaningful and structured questions.
5. **Question Generation**
   - Takes the identified topics and generates **contextual questions**.
6. **Question Answering with Gemini LLM**
   - Uses Gemini LLM to provide **answers based on the original content**.
   - Ensures responses are relevant to the extracted text.
7. **Validation (Accuracy & Reliability)**
   - Compares the **answers generated from the summary vs. the original text**.
   - Uses **embeddings (via gemini) and similarity measures** to check accuracy.
   - Ensures high-quality and consistent responses.
   - **Streamlit:** (If included) Provides a UI for interaction.

6. IMPLEMENTATION

6.1. Importing Libraries

```
import os
import json
import requests
import tempfile
import nltk
import time
import re
from pyzotero import zotero
import fitz  # PyMuPDF
from langchain.agents import initialize_agent, Tool
from langchain.schema import Document
import gensim
import pandas as pd
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from gensim import corpora
from gensim.models import LdaModel
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
import random
from langchain_google_genai import GoogleGenerativeAI, GoogleGenerativeAIEmbeddings
import streamlit as st
```

6.2 Fetch PDF from Zotero Extract Text

```
def fetch_pdf_from_zotero():
    items = zot.items()  # Fetch all top-level items
```

```
    if not items:
        st.warning("No items found in Zotero.")
        return None, None
    pdf_items = []
    for item in items: # Skip if the item itself is an attachment (we want parent items)

        if item["data"].get("itemType") == "attachment":
            continue
        try:
            attachments = zot.children(item["key"])# Fetch attachments for this item
            if any(att["data"]["contentType"] == "application/pdf" for att in
attachments):# Check if any attachment is a PDF
                pdf_items.append(item)
        except zotero_errors.UnsupportedParams:# Skip items that don't support
/children (e.g., non-attachments)
            continue
    if not pdf_items:
        st.warning("No items with PDF attachments found in Zotero.")
        return None, None
    titles = [item["data"]["title"] for item in pdf_items]
    keys = [item["key"] for item in pdf_items]
    return titles, keys
def download_pdf(item_key):
    attachments = zot.children(item_key)
    for attachment in attachments:
        if attachment["data"]["contentType"] == "application/pdf":
            pdf_url = attachment["links"].get("enclosure", {}).get("href")
            if not pdf_url:
                continue
            try:
                response = requests.get(pdf_url, headers={"Zotero-API-Key":
ZOTERO_API_KEY}, stream=True)
                response.raise_for_status()
```

```
            with tempfile.NamedTemporaryFile(delete=False, suffix='.pdf') as temp_pdf:
                for chunk in response.iter_content(chunk_size=8192):
                    temp_pdf.write(chunk)
                return temp_pdf.name
        except Exception as e:
            st.error(f"Error downloading PDF: {e}")
            return None
    st.error("No PDF attachment found for this item.")
    return None
def extract_text_from_pdf(pdf_path):
    try:
        with fitz.open(pdf_path) as pdf:
            return " ".join(page.get_text() for page in pdf)
    except Exception as e:
        st.error(f"Error reading PDF: {e}")
        return ""
```

6.3 SUMMARIZATION AGENT

```
def summarize_text(input_text, llm):
    summarization_prompt = """
    You are an AI assistant tasked with creating concise and informative summaries of research papers for researchers in related fields. Your audience is researchers seeking to quickly understand the core contributions of a paper. Focus on the most important aspects and novelty.

    **Instructions:**
    - Analyze the research paper text provided below.
    - Generate a summary capturing the main points and findings.
    - Ensure the summary is concise, accurate, and objective, written in your own words.
    - Do not include personal opinions, direct quotes, or assume missing information.
    - Use clear, professional language suitable for researchers.
```

```
    - If the text is incomplete, summarize only what is present.
    1. **Title & Objective**: Briefly state the document's title (if available) and its main research question or objective.
    2. **Key Findings**: Summarize the most important results or conclusions.
    3. **Methodology**: Explain how the research was conducted (e.g., experiments, models, datasets used).
    4. **Main Arguments & Evidence**: Highlight the core arguments and supporting evidence.
    5. **Limitations & Challenges**: Identify any limitations, constraints, or challenges noted in the document.
    6. **Influencing Factors**: Mention any external or internal factors affecting the results.
    7. **Implications & Applications**: Discuss how the findings impact the field and potential real-world applications.
    8. **Conclusion & Future Directions**: Summarize the final conclusions and any proposed future research.
    Text to summarize: {input_text}
    Summary:
    """
    full_prompt = summarization_prompt.format(input_text=input_text)
    try:
        response = llm.invoke(full_prompt)
        return response.strip()
    except Exception as e:
        return f"❌ Exception occurred during summarization: {str(e)}"

summarization_tool = Tool(
    name="Summarization Tool",
    func=lambda text: summarize_text(text, llm),
    description="Summarizes academic papers concisely using the provided chunk text."
)
summarization_agent = initialize_agent(
```

```
    tools=[summarization_tool],
    llm=llm,
    agent_type="zero-shot-react-description",
    verbose=False,
    max_iterations=1000,
    max_execution_time=120,
    handle_parsing_errors=True
)
```

6.4 Q&A AGENT

```
def generate_questions(topics):
    prompt = f"""
    You are an AI assistant tasked with generating thoughtful questions based on topics extracted from a research paper.
    Using the following topics, generate exactly three questions that a researcher might ask to explore the content further.
    Topics:
    {', '.join(topics)}
        try:
        response = llm.invoke(prompt)
        return response.strip()
    except Exception as e:
        return f"Error generating questions: {e}"

question_generator_tool = Tool(
    name="Question Generator",
    func=generate_questions,
    description="Generates three thoughtful questions based on provided topics extracted from a document."
)

def summary_content_answer_tool_func(input_json):
    try:
        data = json.loads(input_json)
```

```
        question = data["question"]
        summary_content = data["summary_content"]
        return answer_with_summary_content(question, summary_content)
    except Exception as e:
        return f"Error in summary_content_answer_tool: {e}"

summary_content_answer_tool = Tool(
    name="Summary Content Answer",
    func=summary_content_answer_tool_func,
    description="Answers a question using the provided summarized content.
Provide a JSON with keys 'question' and 'summary_content'."
)

def original_content_answer_tool_func(input_text):
    try:
        parts = input_text.split(" | ")
        question = parts[0].replace("question: ", "").strip()
        original_content = parts[1].replace("content: ", "").strip()
        return answer_with_original_content(question, original_content)
    except Exception as e:
        return f"Error in original_content_answer_tool: {e}"

original_content_answer_tool = Tool(
    name="Original Content Answer",
    func=original_content_answer_tool_func,
    description="Answers a question using the provided original content. Provide
input as 'question: <question> | content: <original_content>'."
)
agent2 = initialize_agent(
    tools=[question_generator_tool, summary_content_answer_tool,
original_content_answer_tool],
    llm=llm,
    agent_type="zero-shot-react-description",
    verbose=False,
```

```
    max_iterations=1000,
    max_execution_time=120,
    handle_parsing_errors=True
)
```

6.5 VALIDATION AGENT

```
def validate_summary(original_data, summary):
    original_keywords = original_data["keywords"]
    keyword_overlap = len(set(original_keywords) & set(summary.split())) / len(original_keywords) if original_keywords else 0
    original_topics = original_data["topics"]
    topic_overlap = sum(len(set(orig.split()) & set(summary.split())) / len(set(orig.split())) for orig in original_topics) / len(original_topics) if original_topics else 0
    result = {
        "keyword_overlap": round(keyword_overlap, 2),
        "topic_overlap": round(topic_overlap, 2),
        "valid": keyword_overlap > 0.5 and topic_overlap > 0.5
    }
    return json.dumps(result)

def validate_answers(summarized_answers, original_answers):
    summarized_embeddings = [embed_content(ans) for ans in summarized_answers if ans.strip()]
    original_embeddings = [embed_content(ans) for ans in original_answers if ans.strip()]
    summarized_embeddings = [e for e in summarized_embeddings if e is not None]
    original_embeddings = [e for e in original_embeddings if e is not None]
    if not summarized_embeddings or not original_embeddings:
        return json.dumps({"similarity": 0.0})
    similarity_matrix = cosine_similarity(np.array(summarized_embeddings), np.array(original_embeddings))
    avg_similarity = np.mean(similarity_matrix)
```

```
    return json.dumps({"similarity": round(avg_similarity, 2)})

summary_validation_tool = Tool(
    name="Summary Validation",
    func=lambda inputs: validate_summary(
        json.loads(inputs)["original_data"] if isinstance(inputs, str) else inputs["original_data"],
        json.loads(inputs)["summary"] if isinstance(inputs, str) else inputs["summary"]
    ),
    description="Validates if the summary retains key topics and keywords from the original text."
)

qa_validation_tool = Tool(
    name="Q&A Validation",
    func=lambda inputs: validate_answers(
        json.loads(inputs)["summarized_answers"] if isinstance(inputs, str) else inputs["summarized_answers"],
        json.loads(inputs)["original_answers"] if isinstance(inputs, str) else inputs["original_answers"]
    ),
    description="Validates Q&A responses by comparing summarized and original answers using embeddings."
)

validation_agent = initialize_agent(
    tools=[summary_validation_tool, qa_validation_tool],
    llm=llm,
    agent_type="zero-shot-react-description",
    verbose=False,
    handle_parsing_errors=True
)
```

7. OUTPUT SCREENS

7.1 Main Output Screen

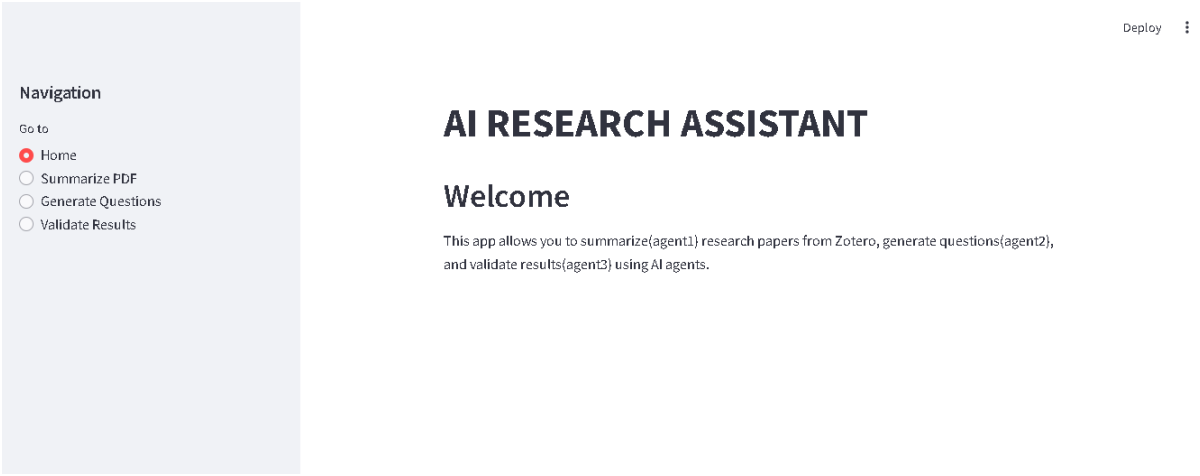


FIG 7.1: MAIN OUTPUT SCREEN

7.2 Summarization Output

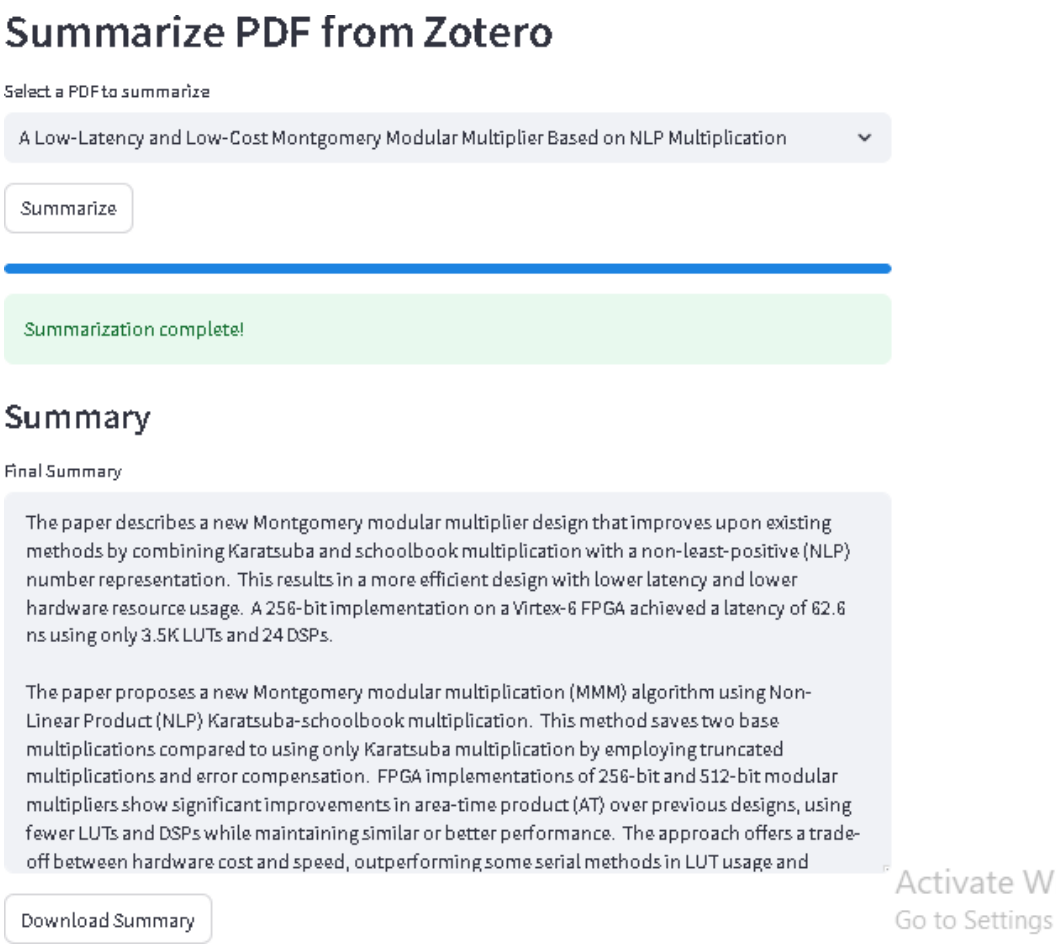


FIG 7.2: SUMMARY OUTPUT

7.3 Q&A AGENT
7.3.1 Generated Questions

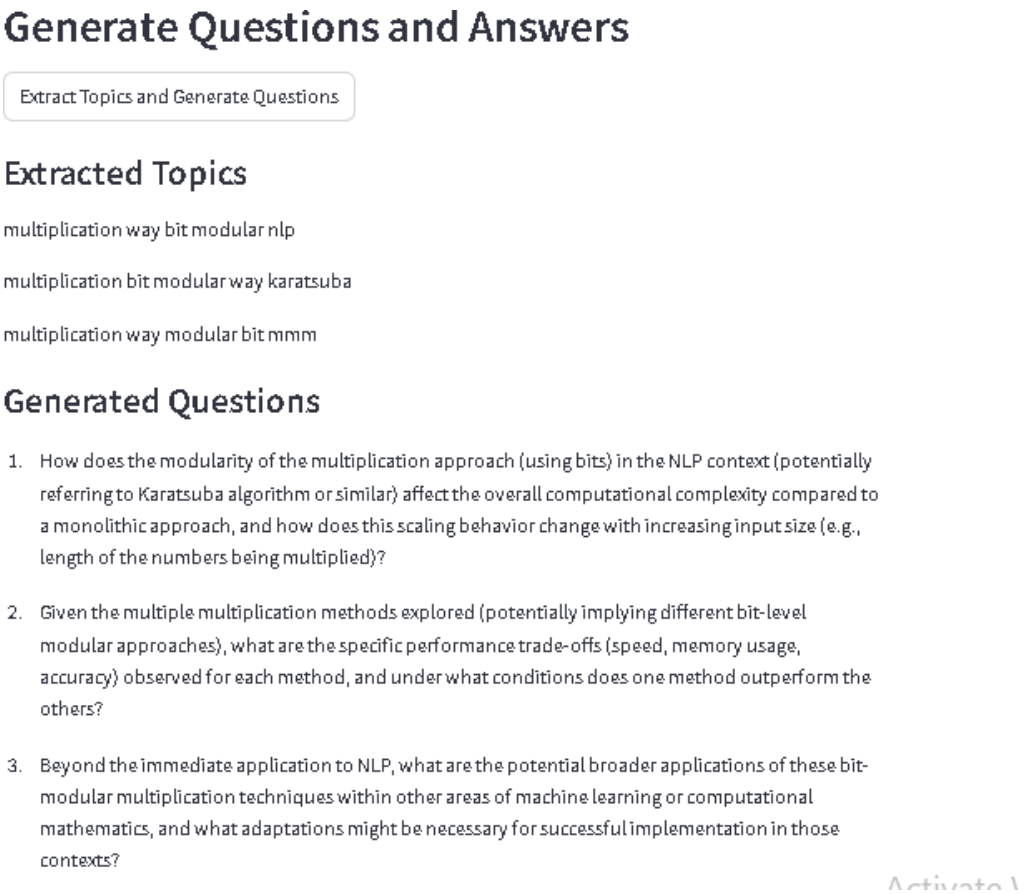
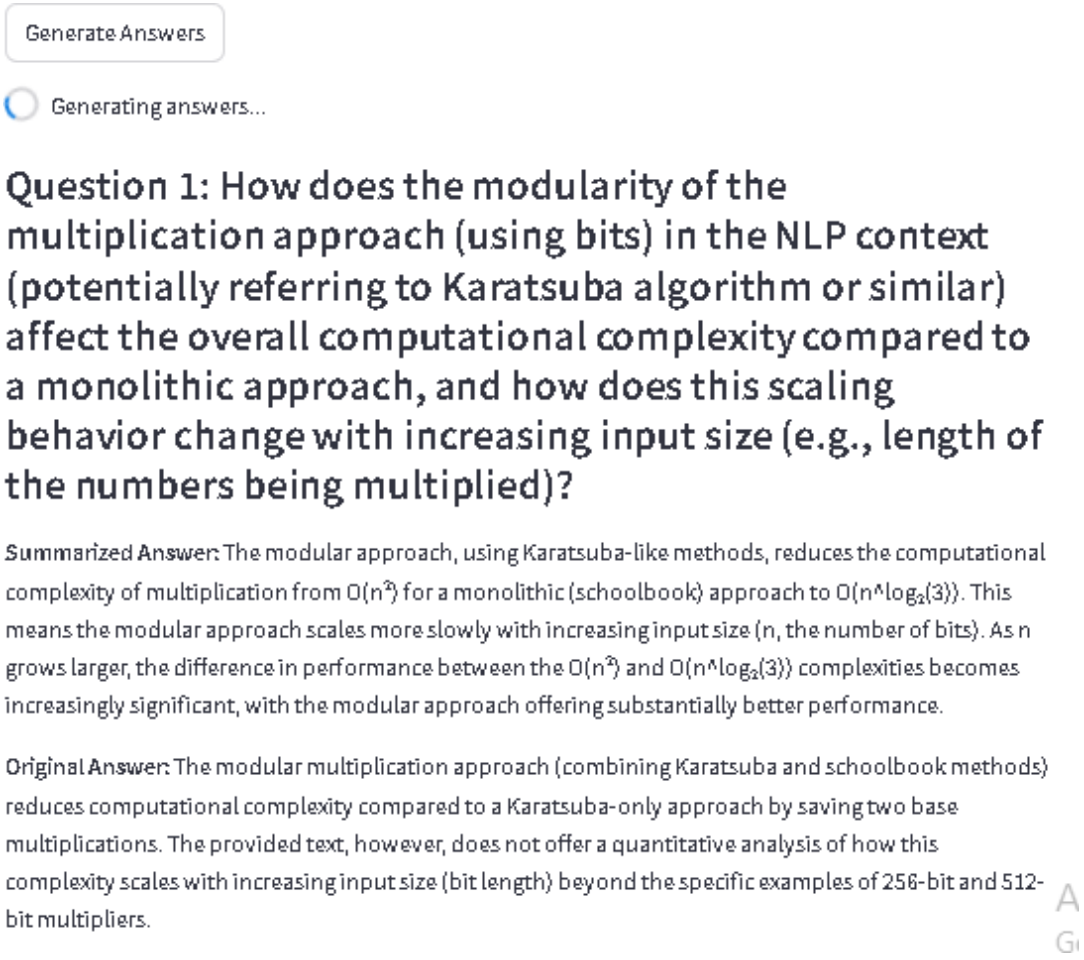


FIG 7.3.1: GENERATED QUESTIONS

7.3.2. Generated Answers



Question 2: Given the multiple multiplication methods explored (potentially implying different bit-level modular approaches), what are the specific performance trade-offs (speed, memory usage, accuracy) observed for each method, and under what conditions does one method outperform the others?

Summarized Answer: The provided summary does not contain enough information to answer the question. While it describes a new Montgomery modular multiplication method that improves upon existing methods, it does not provide a comparison of the performance trade-offs (speed, memory usage, accuracy) between multiple multiplication methods. Therefore, it's impossible to determine under what conditions one method outperforms others based on this summary.

Original Answer: The provided text only describes a modular multiplication method combining Karatsuba and schoolbook multiplication within a Montgomery modular multiplication framework using a non-least positive form (NLP). It states this method saves two base multiplications compared to Karatsuba-only designs. Performance data (latency and resource usage) is given only for this specific method (62.6 ns and 3.5K LUTs/24 DSPs for 256-bit on a Virtex-6 FPGA), but no other methods are described in sufficient detail to allow a comparison of performance trade-offs. Therefore, a complete answer to the question about performance trade-offs between multiple methods is not possible based on the provided text.

Question 3: Given the observed variations in performance across different multiplication methods, can a predictive model be developed to determine the optimal multiplication strategy based on input characteristics (size, type, etc.) and available computational resources?

Summarized Answer: Based on the provided summary, the development of a predictive model to determine the optimal multiplication strategy is plausible due to the observed performance differences between various methods. However, the summary lacks the quantitative data necessary to definitively confirm the feasibility of such a model. More information on the performance characteristics and relationships between input characteristics and computational resources would be needed.

Original Answer: The provided text does not contain sufficient information to determine whether a predictive model can be developed to determine the optimal multiplication strategy. The research presented focuses on a specific method and its performance, not on a comparative analysis across various methods and input characteristics necessary to build such a model.

FIG 7.3.2: GENERATED ANSWERS

7.4 VALIDATION AGENT3

7.4.1. SUMMARY VALIDATION

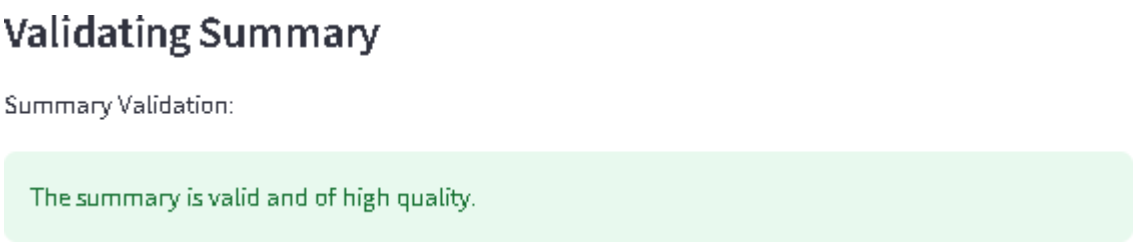


FIG 7.4.1: SUMMARY VALIDATION

7.4.2. Q&A VALIDATION

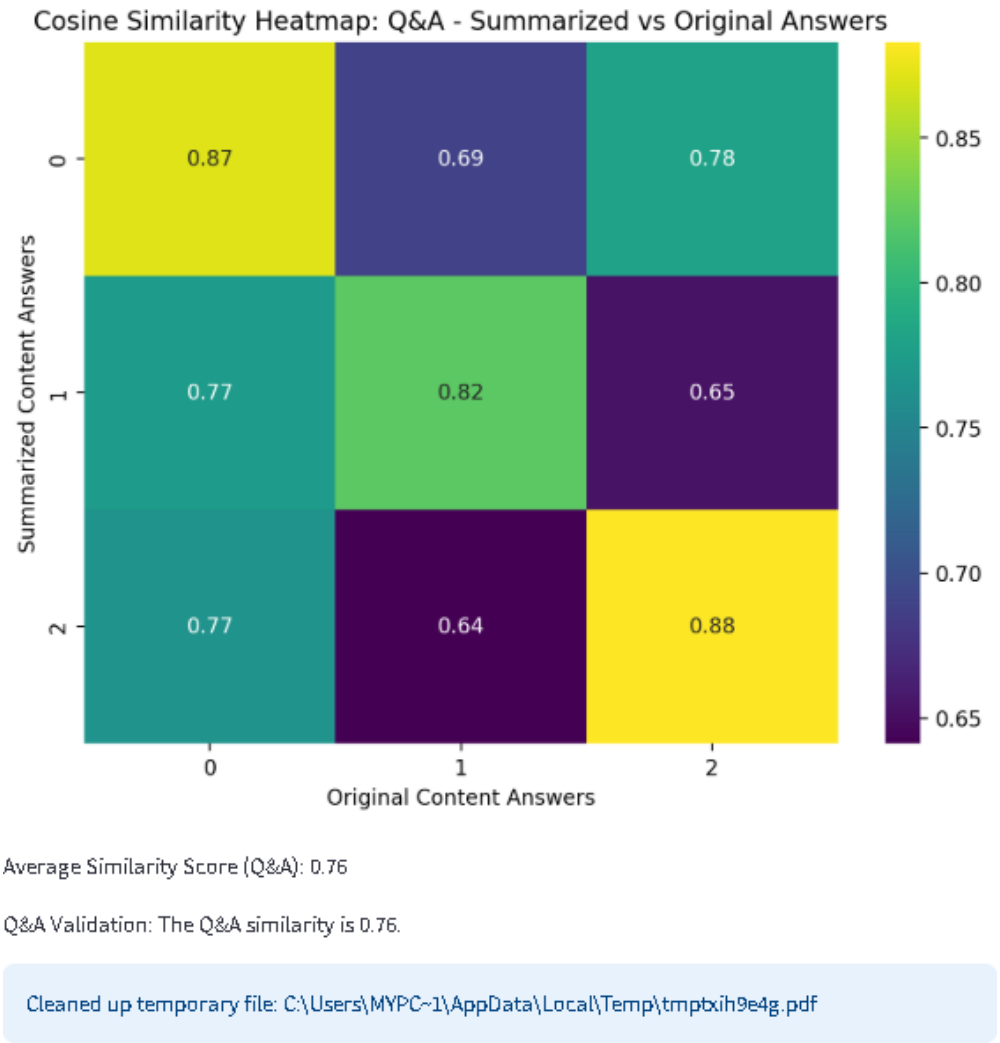


FIG 7.4.2: Q&A VALIDATION

8. CONCLUSION

Project Overview and Purpose

The AI Research Assistant developed in this project integrates advanced AI and NLP technologies with the Zotero API to streamline academic research workflows. Designed to assist researchers in efficiently processing scholarly papers, the tool automates summarization, question generation, and result validation, reducing the time and effort required to extract meaningful insights from complex documents.

Key Achievements

The system successfully leverages Google's Generative AI models for concise and accurate summarization, topic extraction, and question generation, while a Streamlit interface provides an intuitive user experience. By fetching PDFs from Zotero, summarizing content, and validating outputs against original texts, the assistant ensures both usability and reliability, demonstrating a robust end-to-end solution for research support.

Technical Strengths

The integration of LangChain agents, TF-IDF keyword extraction, LDA topic modeling, and cosine similarity-based validation showcases a sophisticated blend of modern NLP techniques. These components enable the assistant to produce high-quality summaries, generate relevant research questions, and assess consistency between summarized and original content, making it a valuable asset for academic exploration.

Challenges and Limitations

Despite its capabilities, the tool faces challenges such as potential API rate limits, errors in PDF text extraction, and reliance on well-structured input data. These limitations occasionally impact performance, particularly with large or poorly formatted documents, underscoring the need for robust error handling and broader compatibility in future iterations.

9. FUTURE ENHANCEMENTS

1. Advanced Summarization Techniques

- Adaptive Summarization Models: Use fine-tuned LLMs that adapt to user-specific research domains for more context-aware summarization.
- Multi-document Summarization: Extend the system to process multiple documents simultaneously, consolidating insights across various sources.
- Customizable Summarization Levels: Allow users to adjust the level of detail in summaries (e.g., brief highlights vs. in-depth summaries).

2. Enhanced Question-Answering Capabilities

- Interactive Q&A System: Implement a conversational interface where users can ask follow-up questions based on the summaries.
- Multilingual Support: Incorporate language translation capabilities to handle research articles in multiple languages.
- Topic-Specific Q&A Models: Train models to specialize in specific fields, such as medicine, engineering, or social sciences, ensuring more accurate answers.

3. Validation and Quality Assurance

- Contextual Validation: Expand the validation process to include deeper semantic analysis, such as detecting paraphrasing or subtle inaccuracies in the summaries.
- User Feedback Loop: Implement a mechanism to collect user feedback on summaries and answers, using it to improve the system iteratively.
- Plagiarism Detection: Add a module to check if the summaries or extracted content align with ethical research practices and avoid plagiarized information.

4. Visualization and Interaction

- Dynamic Visualizations: Replace heatmaps with more interactive tools, such as network graphs or topic flow diagrams, for better understanding of content relationships.
- Dashboard Integration: Create an intuitive dashboard to allow users to monitor progress, view summaries, and analyze insights.
- Custom Visualizations: Give users control over the type of visualizations generated (e.g., bar graphs, word clouds).

REFERENCES

[1]. McKinney, W., & Heuser, S. (n.d.). Pyzotero: Python client for the Zotero API. GitHub Repository. Retrieved from https://github.com/urschrei/pyzotero

[2]. LangChain. (2023). LangChain: A framework for building applications with language models. GitHub Repository. Retrieved from https://github.com/langchain-ai/langchain

[3]. Google. (2024). Google Generative AI API Documentation. Retrieved from https://developers.google.com/generative-ai

[4]. Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media. Retrieved from https://www.nltk.org/

[5]. Řehůřek, R., & Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 45-50. Retrieved from https://radimrehurek.com/gensim/

[6]. Artifex Software. (n.d.). PyMuPDF: Python bindings for MuPDF. GitHub Repository. Retrieved from https://github.com/pymupdf/PyMuPDF

[7]. Streamlit Inc. (2023). Streamlit: A faster way to build and share data apps. Retrieved from https://streamlit.io/

[8]. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830. Retrieved from https://scikit-learn.org/

GITHUB LINK:

https://github.com/DARLING-7/AI_RESEARCH_ASSISTANT