Personal AI Assistant Using LangGraph and LangChain

Vishal Vardhan Adepu — 334003567

November 3, 2024

Abstract

This report describes the design and implementation of a personal AI assistant built using LangGraph and LangChain. The assistant provides functionalities such as sending emails, reading and answering questions from PDF files, scheduling meetings, and searching the Internet. Privacy concerns are mitigated through a local LLM model to ensure that private information is handled securely. This report details the features, implementation, challenges, and future work of the project.

1 Problem Statement

The goal of this project is to create a personal AI assistant that can perform the following tasks:

- Write and send emails on behalf of the user (1pt)
- Read multiple PDF files and answer questions (1pt)
- Schedule meetings for the user (2pt)
- Search the Internet (2pt)
- Ask clarifying questions when uncertain (2pt)

Key Requirement: The assistant must not leak private information and should use a local LLM for sensitive data. Public LLM APIs may be used for non-private data.

2 Implementation

The assistant is implemented as a command-line application using the `click` library in Python. The following components and tools were used:

2.1 Workflow

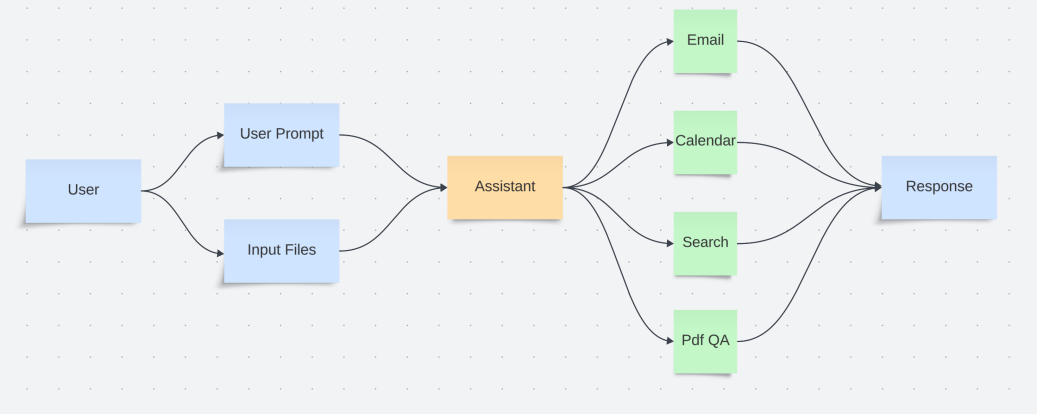

Figure 1: Workflow diagram of the personal AI assistant

2.2 Features and Tools

- **Gmail Tool:** Implemented using the `composio` library for composing and sending emails.
- **Search Tool:** Uses the DuckDuckGo API from LangChain to perform Internet searches.
- **PDF QA Tool:** Built using LangChain and backed by a Chroma vector database for efficient question answering. The assistant can also answer questions about previously uploaded PDFs, as the data is persisted in the local file system using ChromaDB.
- **Calendar Tool:** Also implemented using the `composio` library for scheduling meetings.

- **Privacy Checks:** Utilized the Mistral-NeMo 12B model running locally via Ollama. The local LLM first checks for any private information in the user input. If private data is detected, the assistant flags the information and requests user consent before proceeding. If consent is denied, the input is erased from memory.

2.3 Privacy Mechanism

The assistant uses a local LLM model to identify and handle private information securely:

1. User input is analyzed for private information using the local model.
2. If private data is detected, the assistant informs the user and asks for consent to share it with a public LLM.
3. If consent is given, the information is shared; otherwise, the input is erased and the user is prompted again.

3 Clarifying Questions

The assistant asks for additional details when prompts are vague. For example, in the Gmail feature demo, the assistant requests clarification if the input lacks necessary information. This behavior is consistent across all tools.

4 Screenshots

4.1 Gmail Feature

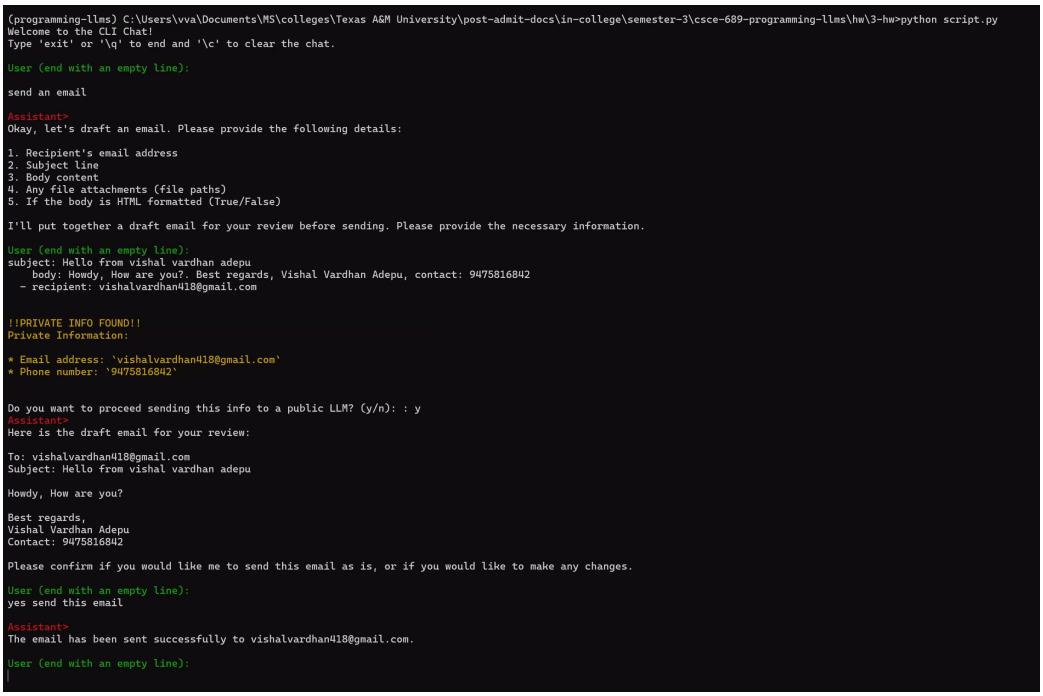

Figure 2: Screenshot showing the command-line execution of the Gmail feature, including privacy checks and user confirmation

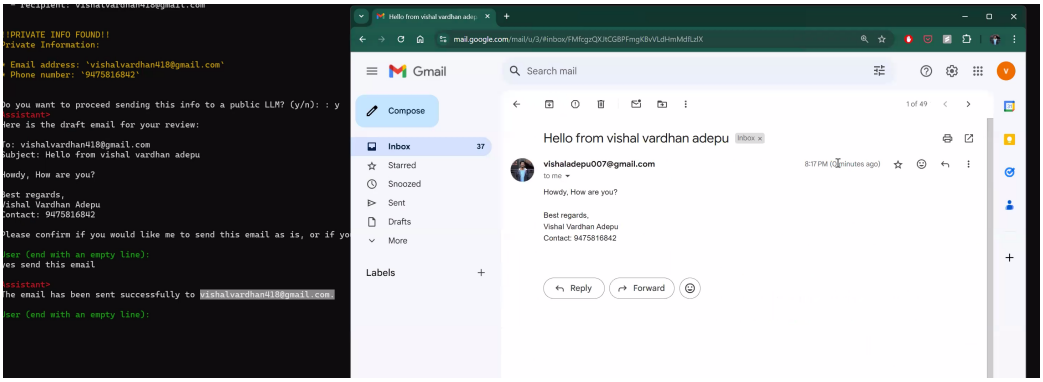

Figure 3: Screenshot showing the sent email as received in Gmail

4.2 Calendar Feature

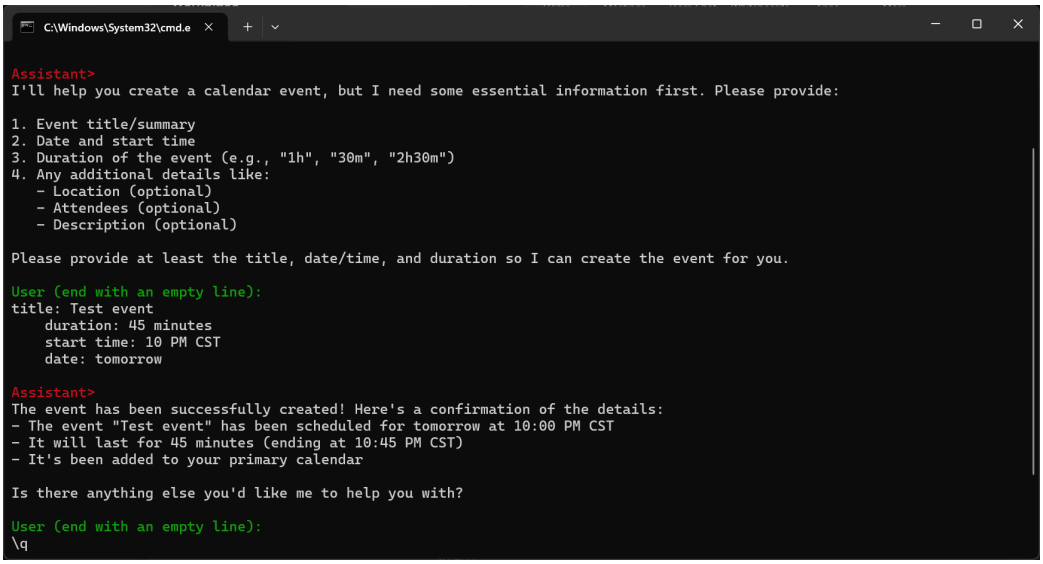

Figure 4: Screenshot showing the command-line execution of the Calendar feature, where an event is created successfully and details are confirmed with the user

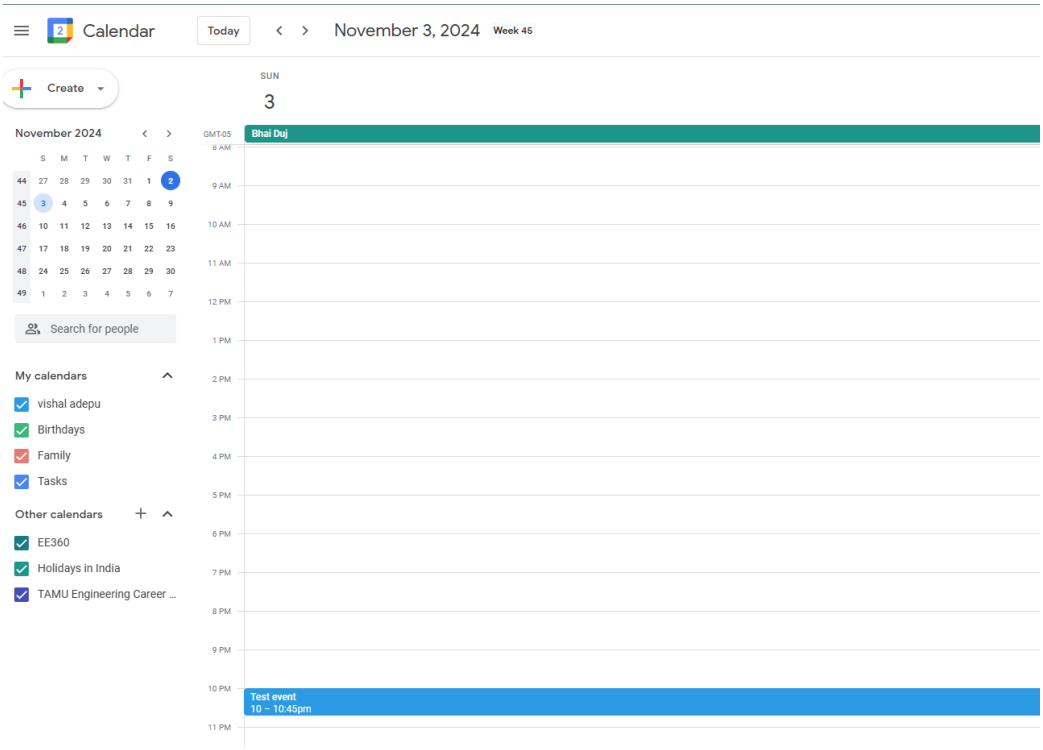

Figure 5: Screenshot showing the created event in Google Calendar with the correct date and time

4.3 Search Feature

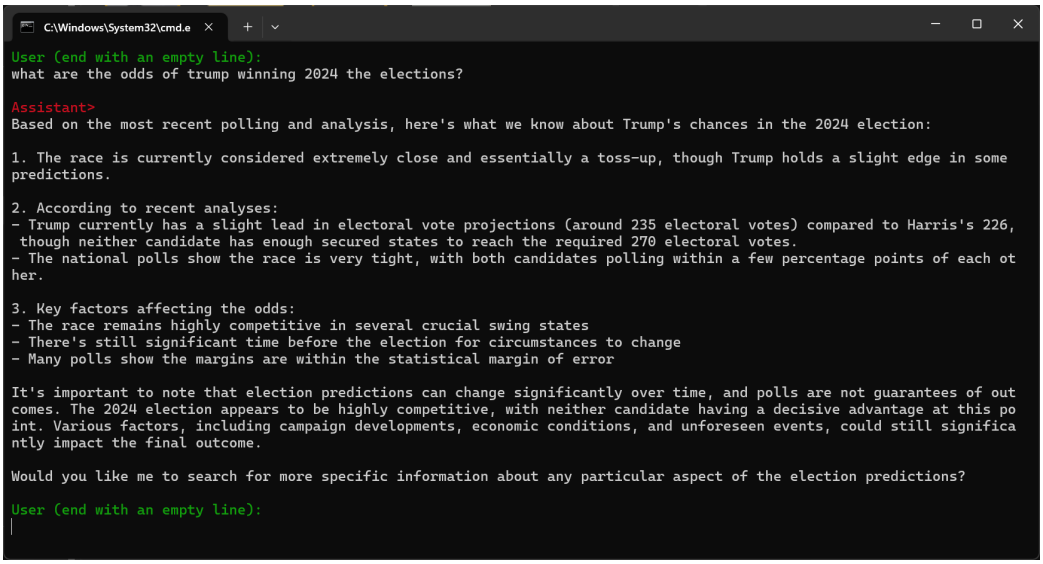

Figure 6: Screenshot demonstrating a search query execution and the assistant's response

4.4 PDF QA Feature

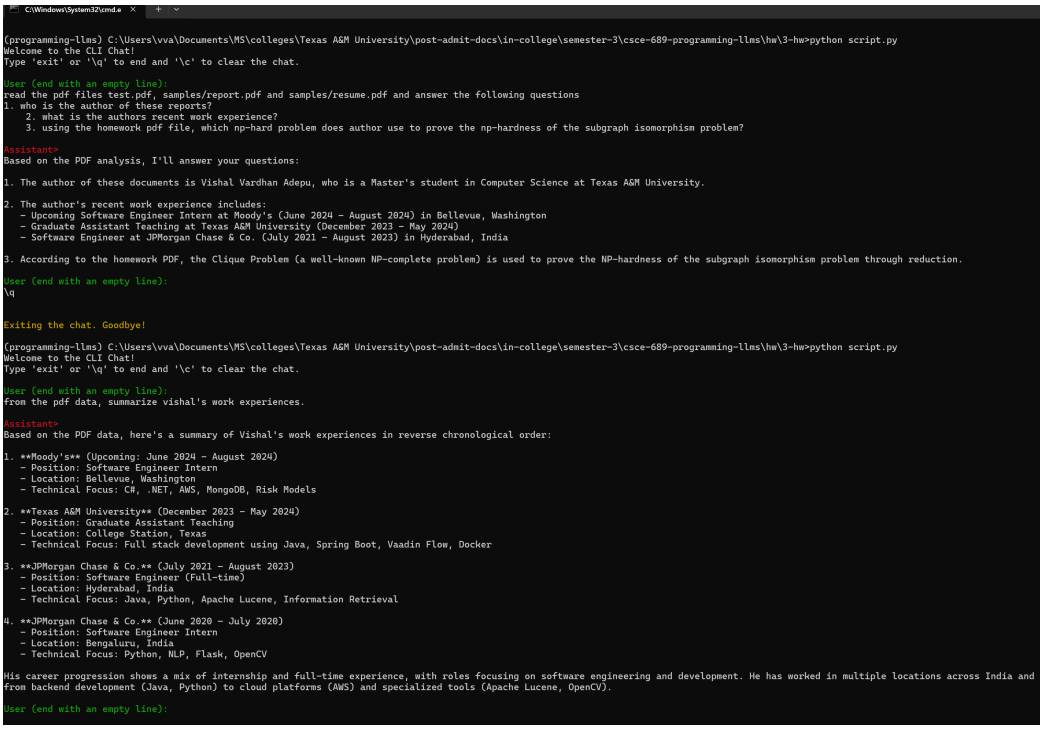

Figure 7: Screenshot showing the assistant answering questions from previously uploaded PDF files, demonstrating data persistence using ChromaDB

5 Environment Variables

The assistant's behavior can be modified using environment variables:

- **FULLY_LOCAL:** When set to 1, the assistant uses only the local LLM. Default is 0.
- **SKIP_PRIVACY_CHECK:** When set to 1, privacy checks are skipped, and input is directly sent to a public LLM. Default is 0.
- **CONTEXT_HISTORY_LEN:** Adjusts the memory context length. A higher value increases context size but also API call costs. Default is 20.

6 Code and Demo

The project code is available in a ZIP file, which includes a `README.md` with instructions for running the assistant. The demo video can be viewed at: [Video Link]. The demo showcases the assistant's capabilities, including email composition, meeting scheduling, PDF question answering, and Internet search.

7 Challenges Faced

- **Learning Curve:** Understanding LangChain and LangGraph was difficult due to limited documentation and a steep learning curve.
- **Prompt Engineering:** Multiple iterations were required to fine-tune prompts and achieve desired LLM behavior.
- **PDF QA Tool:** Linking LangChain with a vector database involved grasping complex concepts like vector embeddings and database integration. Additionally, ensuring data persistence for answering questions about previously uploaded PDFs was challenging.
- **Memory and State Management:** Managing context and memory in LangChain was challenging because the documentation was not clear about library functions.

8 Conclusion

This project successfully implements a personal AI assistant with robust privacy mechanisms and efficient functionality. The assistant can handle tasks like sending emails, scheduling meetings, searching the Internet, and answering questions from PDFs with data persistence. Future work includes enhancing memory management and exploring more advanced prompt engineering techniques.