NORTH SOUTH UNIVERSITY

Department of Electrical & Computer Engineering (ECE)

Semester: Spring 2025

Course: CSE482 Section: 2

Faculty: Mirza Mohammad Lutfe Elahi

GROUP NO: 02

NAME	STUDENT ID
Farjana Rahman Samia	2211754042
Naima Zaman Roshni	2212628042
Shuvra Saha	2122468642

Submission Date: 22/04/2025

BookBuddy: Personalised Book Suggestions

Abstract:

Bookbuddy is an AI-powered chatbot designed to recommend books based on user queries. It processes PDF files of books, extracts and splits the text into smaller chunks, and converts them into vector embeddings using a sentence transformer model. These embeddings are stored in a FAISS index for efficient retrieval. When a user asks for recommendations, **Bookbuddy** retrieves the most relevant content and uses a selected language model (like **LLaMA 3** or **DeepSeek** or **Qwen2.5)** to generate personalized suggestions. The chatbot is built with **Streamlit**, providing an interactive and user-friendly interface. You can find project code here: https://github.com/NaimaRoshni/BookBuddy

Introduction:

In today's digital age, readers often find it difficult to choose the right book from the overwhelming number of options available. Bookbuddy aims to solve this problem by offering intelligent book recommendations through a conversational interface. Instead of relying on traditional recommendation systems that use predefined categories or ratings, Bookbuddy allows users to ask natural language questions and receive context-aware suggestions.

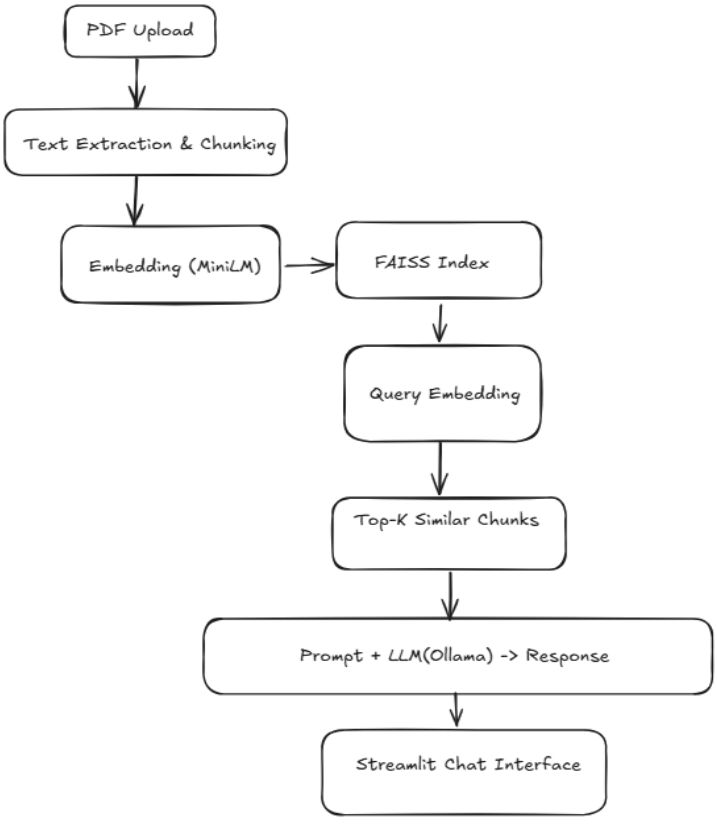
The idea behind Bookbuddy is to make book discovery more interactive and personalized. Users can upload books in PDF format, which are then processed and stored in a searchable format. By combining modern natural language processing techniques with efficient vector search and large language models (LLMs), Bookbuddy can understand user queries and recommend books based on actual content, not just titles or genres.

This project demonstrates how tools like **FAISS, LangChain, Ollama**, and **Streamlit** can work together to build an end-to-end AI-powered recommendation system. It is designed for students,

educators, and book enthusiasts who are looking for smart and meaningful suggestions tailored to their interests.

System Architecture

Bookbuddy is built as an end-to-end AI system that processes book content, stores it for efficient retrieval, and uses large language models to generate book recommendations based on user queries.



BOOKBUDDY ARCHITECTURE

Pipeline Overview

The core of the Bookbuddy system relies on a well-prepared dataset of books. The entire pipeline—from text extraction to embedding and indexing—was carefully designed to ensure high-quality recommendations.

Book Dataset

The core of the Bookbuddy system relies on a well-prepared dataset of books. The entire pipeline—from text extraction to embedding and indexing—was carefully designed to ensure high-quality recommendations.

Sample Dataset:

The Girl with the Dragon Tattoo by Stieg Larsson (2005)
Category: Mystery, Crime Thriller
Summary: A disgraced journalist, Mikael Blomkvist, teams up with Lisbeth Salander, a brilliant but troubled hacker with a dark past, to unravel the decades-old disappearance of Harriet Vanger, the niece of a wealthy and influential industrialist. As they delve deeper into the mystery, they uncover a web of corruption, family secrets, and violence, facing powerful adversaries who will stop at nothing to protect their hidden truths.

Gone Girl by Gillian Flynn (2012)
Category: Psychological Thriller, Mystery
Summary: On their fifth wedding anniversary, Nick Dunne finds his wife, Amy, missing under mysterious circumstances. As the investigation gains media attention, Nick becomes the prime suspect, revealing a complex and toxic marriage filled with manipulation, lies, and betrayals. Shocking twists expose the unsettling truth about Amy's disappearance, blurring the line between victim and villain in this chilling portrait of a marriage unraveling.

The Da Vinci Code by Dan Brown (2003)
Category: Mystery, Conspiracy Thriller
Summary: When a murder occurs inside the Louvre, Harvard symbologist Robert Langdon is called upon to decipher cryptic messages left at the crime scene. Teaming up with cryptologist Sophie Neveu, he embarks on a high-stakes journey across Europe, uncovering religious secrets, hidden codes, and an ancient conspiracy tied to the Catholic Church. As they race against time, Langdon and Neveu must outwit a dangerous enemy determined to protect the truth at any cost.

The Silent Patient by Alex Michaelides (2019)
Category: Psychological Thriller, Mystery
Summary: Renowned painter Alicia Berenson, once vibrant and successful, becomes an enigma after being accused of brutally murdering her husband. Following the crime, she stops speaking entirely. Psychotherapist Theo Faber becomes obsessed with treating her, believing he can unlock the secrets behind her silence. As Theo digs deeper into her past, he finds himself entangled in a dangerous web of deceit, trauma, and shocking revelations.

Text Extraction:

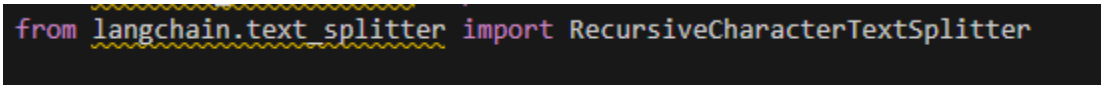
To extract readable text from the PDFs, the project uses the **pdfplumber** library. It scans each page of a PDF and pulls out clean text, ensuring that only meaningful and structured content is passed to the next stage.

```python
# Load PDF and extract text
def load_pdf(pdf_path):
    text = ""
    with pdfplumber.open(pdf_path) as pdf:
        for page in pdf.pages:
            page_text = page.extract_text()
            if page_text:
                text += page_text + "\n"
    return text.strip()
```

Python

Chunking Strategy:

The extracted text is often too long for direct processing, so it is split into smaller, manageable pieces using the



```python
from langchain.text_splitter import RecursiveCharacterTextSplitter
```

- Chunk Size: 500 Characters
- Overlap: 50 Characters

This overlapping strategy helps retain context across adjacent chunks, improving the quality of downstream embeddings and search.

```
# Initialize text splitter
text_splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
```

Embeddding:

Each chunk is converted into a dense vector using the **all-MiniLM-L6-v2** model from Sentence Transformers. This model was chosen for its balance of performance and efficiency in generating semantic embeddings suitable for search and retrieval tasks.

```
# Load embedding model
model = SentenceTransformer("all-MiniLM-L6-v2")
```

Vector Storage with FAISS:

The embeddings are stored in a FAISS index (book_index.faiss) for fast similarity search. FAISS allows quick retrieval of top relevant chunks based on user queries. Additionally, the corresponding text chunks are saved as metadata (metadata.pkl) to reconstruct meaningful context during recommendation.

```
# Ensure FAISS index folder exists
FAISS_PATH = "faiss_index"
if not os.path.exists(FAISS_PATH):
    os.makedirs(FAISS_PATH)
```

Main Function:

After all the process, we run our main function to do all the works together. Here it is:

```python
# Process book PDFs and store in FAISS
def process_pdf(pdf_path):
    text = load_pdf(pdf_path)
    chunks = text_splitter.split_text(text)

    # 🔴 Remove empty chunks
    valid_chunks = [chunk for chunk in chunks if chunk.strip()]
    if not valid_chunks:
        print(f"ERROR: No valid text found in {pdf_path}")
        return

    # Generate embeddings
    embeddings = model.encode(valid_chunks).astype(np.float32)

    # Initialize FAISS index
    index = faiss.IndexFlatL2(embeddings.shape[1])
    index.add(embeddings)

    # Save FAISS index and chunk metadata
    faiss.write_index(index, os.path.join(FAISS_PATH, "book_index.faiss"))

    with open(os.path.join(FAISS_PATH, "metadata.pkl"), "wb") as f:
        pickle.dump(valid_chunks, f)

    print(f"✅ Stored {len(valid_chunks)} book embeddings in FAISS.")
```

```python
# Run pipeline
if __name__ == "__main__":
    process_pdf("booklist.pdf")  # Change this to your actual book file
```

This pipeline enables Bookbuddy to transform unstructured PDF data into a searchable and intelligent knowledge base that

supports personalized and context-aware book recommendations.

Backend Logic:

The main.py script is the core engine behind Bookbuddy's recommendation system. It connects the vector database, embedding model, and language models to generate meaningful responses. Here's what it does:

- **Loads Data:**
  - Reads the FAISS index (book_index.faiss) containing vector embeddings of book chunks.
  - Loads metadata (metadata.pkl) to retrieve original text chunks for context.
- **Embeds User Queries:**
  - Uses the **all-MiniLM-L6-v2** model to embed user queries into vector form, compatible with the FAISS index.
- **Retrieves Relevant Context:**
  - Searches the FAISS index for the top similar chunks to the user query.

- These chunks form the contextual background for the language model to generate accurate responses.

- **Constructs Prompt for LLM:**

  - Combines the retrieved context and user question into a structured prompt using **LangChain's ChatPromptTemplate.**

- **Generates Recommendation:**

  - Sends the prompt to a selected local LLM (via OllamaLLM).
  - Supported models include:
    1. **Llama3.2: latest**
    2. **Deepseek-r1:1.5b**
    3. **Qwen2.5:latest**

- **Returns Output:**

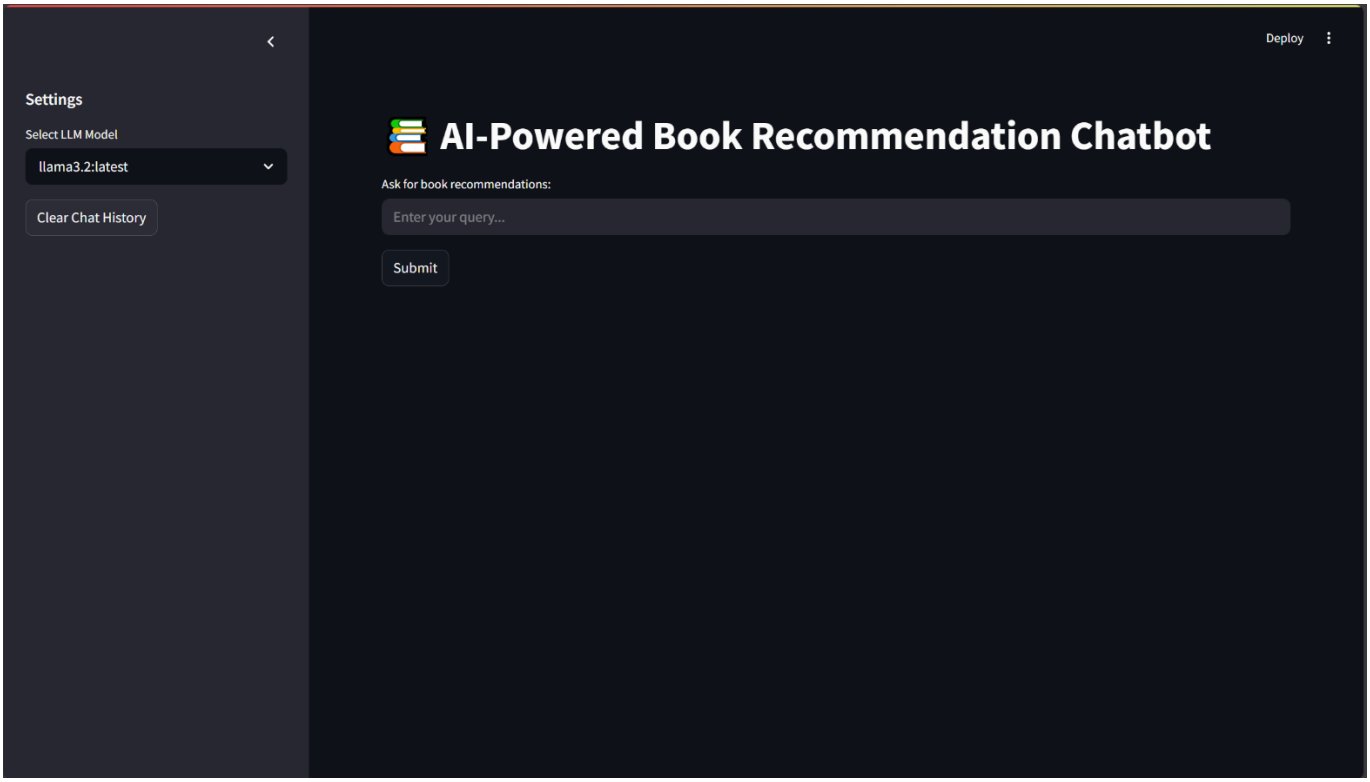
- Uses LangChain's pipeline to process the output and send the final response back to the frontend.

This script effectively links the embedding-based search with modern LLMs to generate context-aware and user-friendly book recommendations.

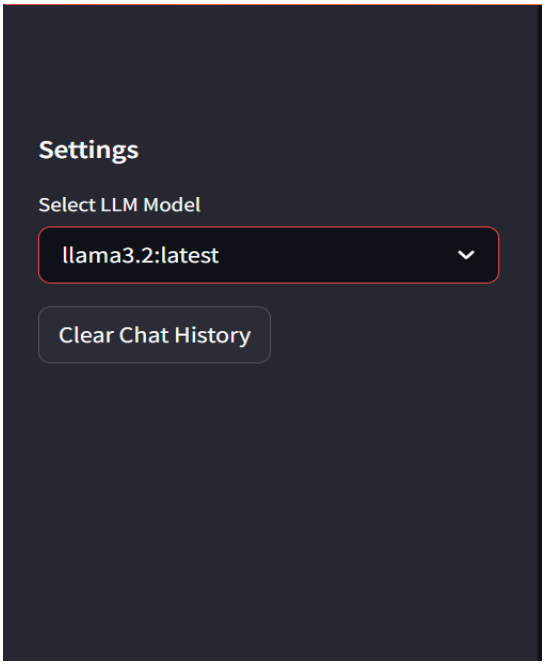
*Frontend – Streamlit Interface*User Features:

The frontend of Bookbuddy is built using **Streamlit**, providing a clean, interactive, and user-friendly interface for accessing the book recommendation system. Here's how it works:

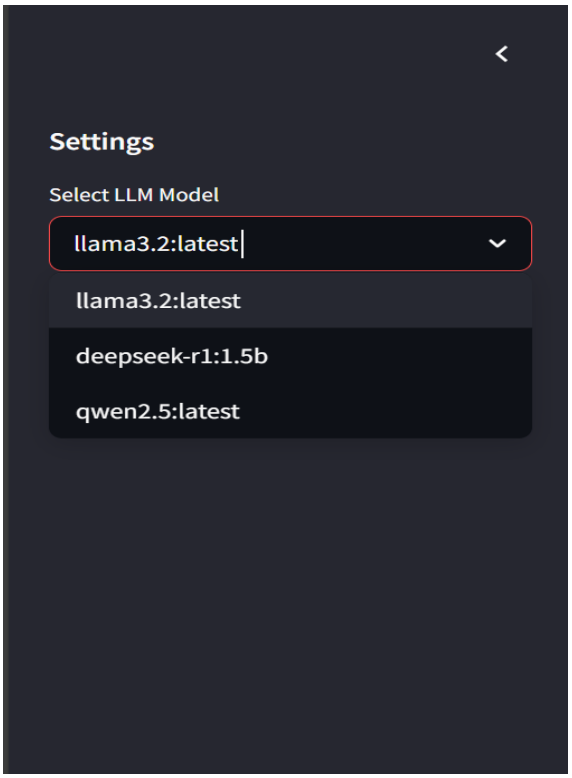
Page Setup:



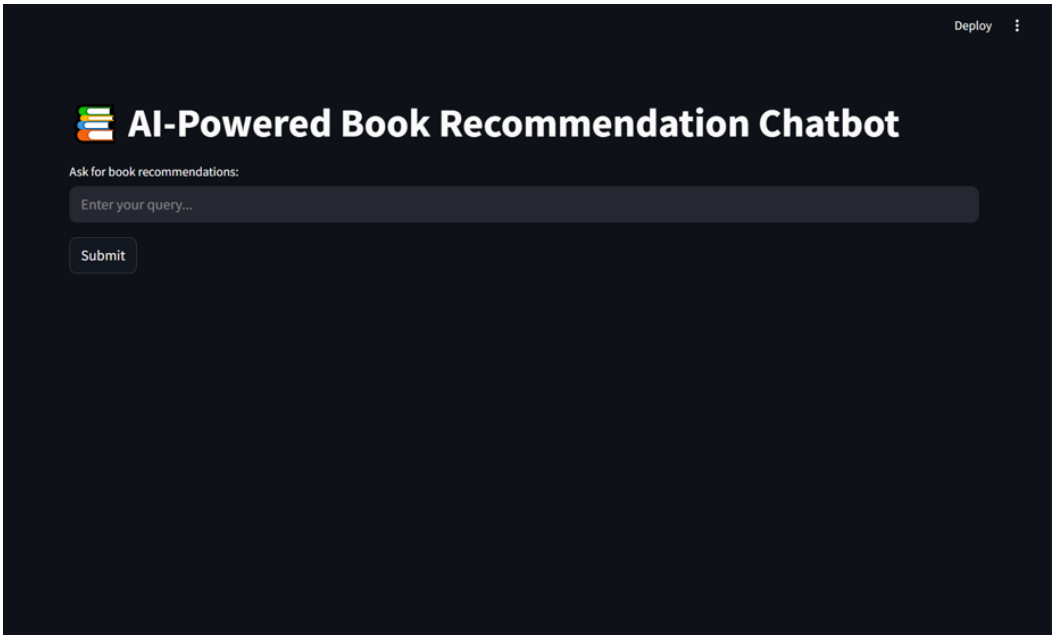
Model Selection (Sidebar):



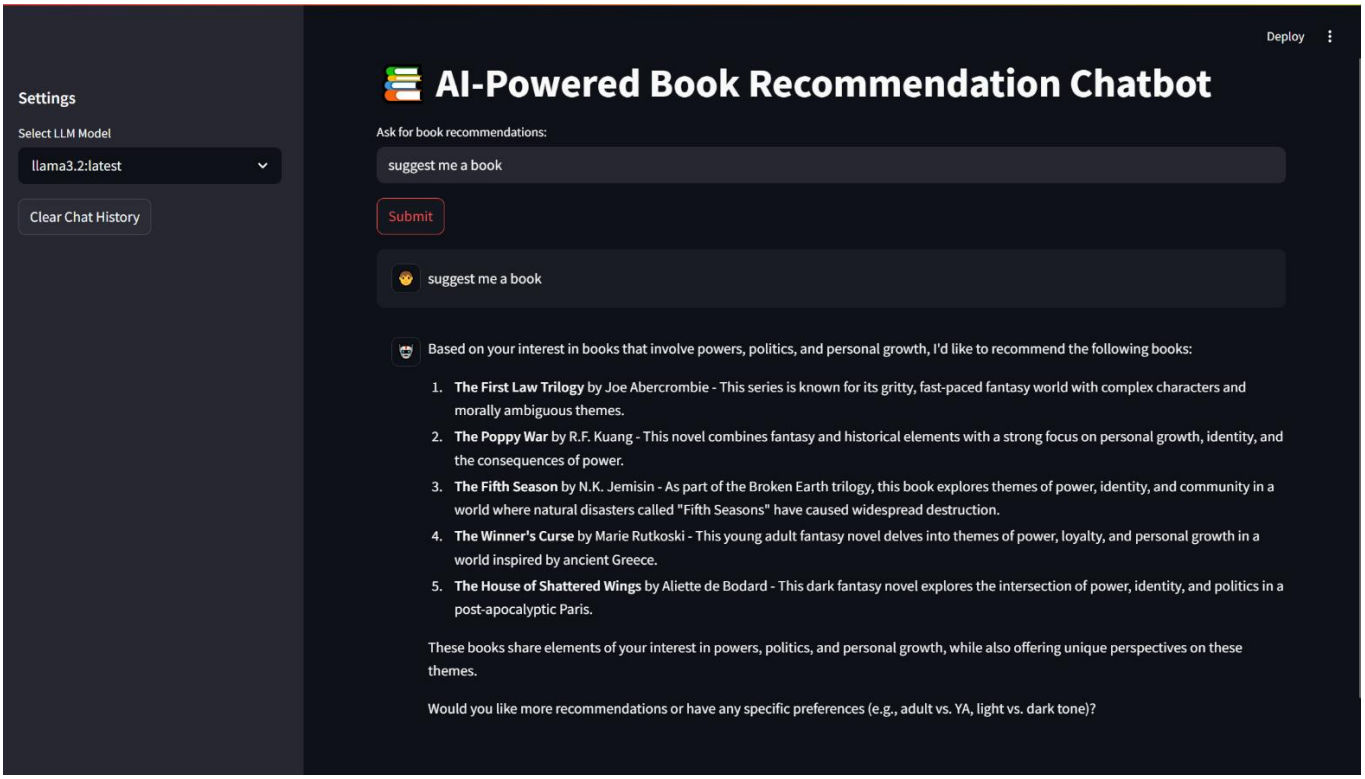
Models:



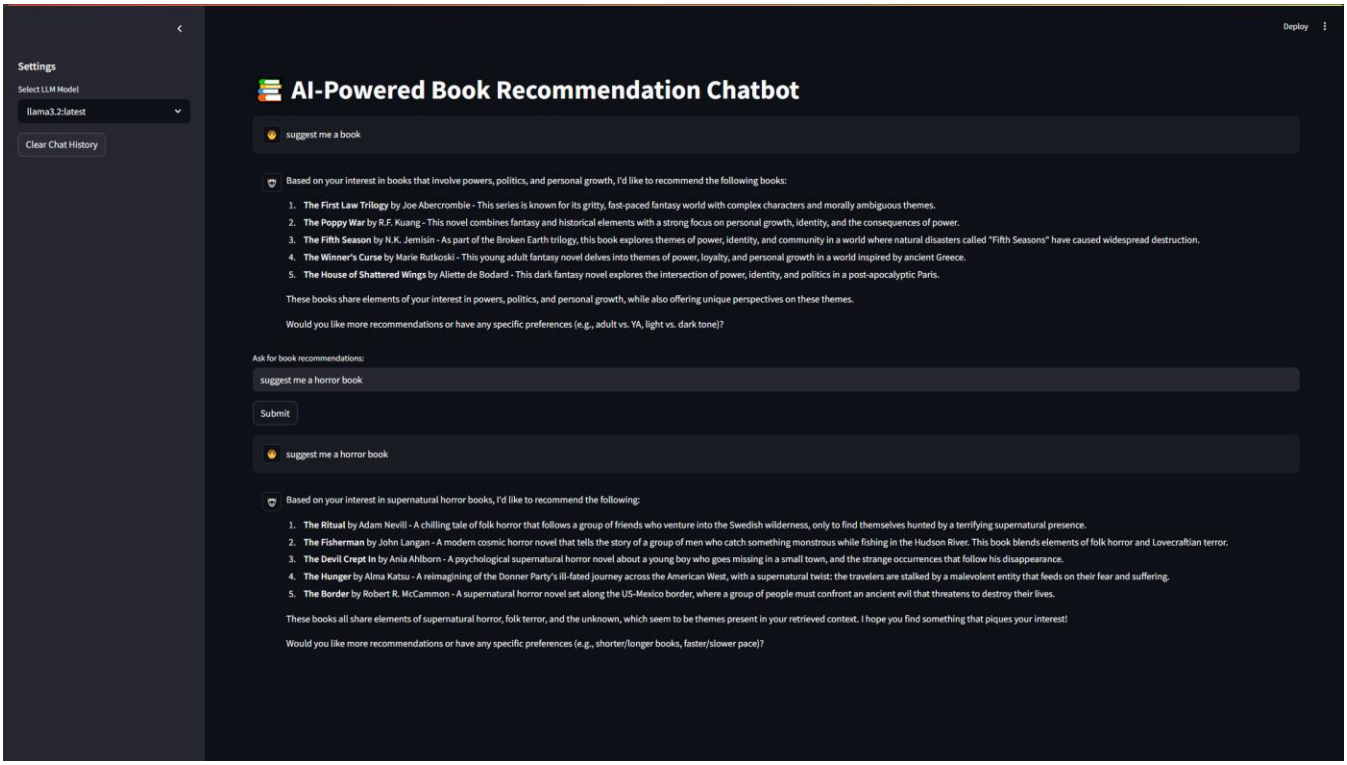
Chat Interface:



Response Display:



Chat History with clear history option:



Technologies Used:

- **Python**
- **Streamlit** – for the user interface
- **LangChain** – for prompt management and chaining components
- **FAISS** – for vector storage and similarity search
- **pdfplumber** – for extracting text from PDFs
- **SentenceTransformers** – for generating text embeddings

- **Ollama** – for running local LLMs
- **LLMs**

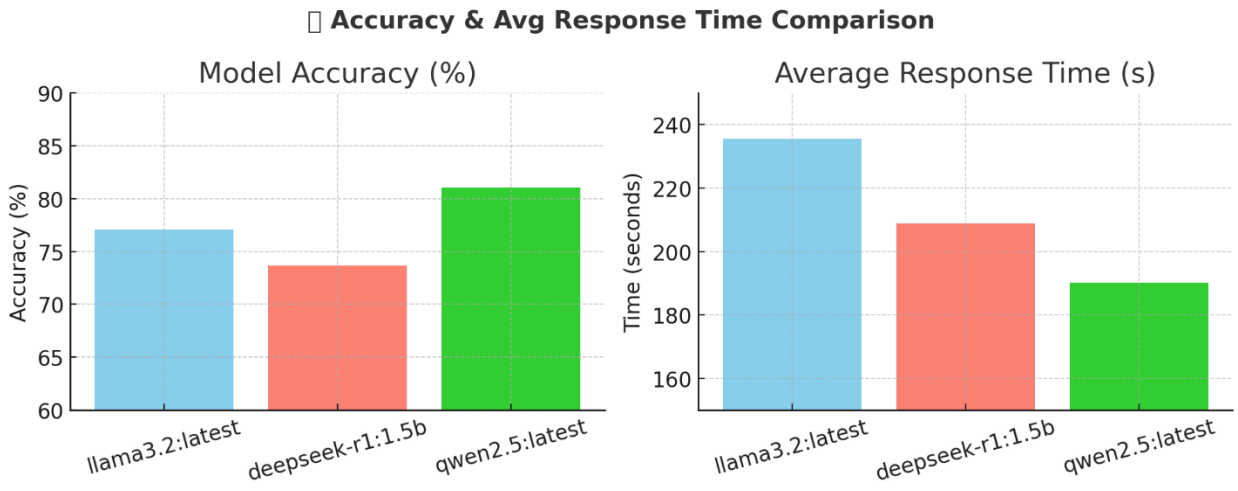
Result:

The performance of three local LLMs—**llama3.2:latest**, **deepseek-r1:1.5b**, and **qwen2.5:latest**—was evaluated using a custom-built book recommendation chatbot. Each model was assessed on accuracy, precision, recall, F1 score, and average response time based on category-specific prompts and a curated book title dataset.

Model Performance Comparison

Model	Accuracy (%)	Precision	Recall	F1 Score	Avg Time (s)
llama3.2:latest	77.08	0.341	0.341	0.341	235.41
deepseek-r1:1.5b	73.68	0.295	0.295	0.295	208.9
qwen2.5:latest	81.05	0.3875	0.3875	0.3875	190.12

Results Comparison:



- **qwen2.5:latest** (7B parameters) delivered the **best overall performance** with **81.05% accuracy**, the **highest F1 score (0.3875)**, and the **fastest response time (190.12s)**—making it ideal for tasks requiring both speed and precision.
- **llama3.2:latest** had solid accuracy (**77.08%**) but was the **slowest** at **235.41s**, making it more suitable for quality-focused applications where latency is less critical.
- **deepseek-r1:1.5b** offered a **balanced but lower performance** across all metrics (**73.68% accuracy**, **208.90s time**), serving as a middle-ground option.

The bar chart visually highlights **qwen2.5's efficiency and effectiveness**, especially given its larger **7B parameter size** compared to the other models.

Challenges & Solutions:

 Challenge: Handling unstructured PDF content.
Solution: Used pdfplumber for accurate text extraction and filtered out empty or noisy chunks.

 Challenge: Ensuring relevant recommendations.
Solution: Combined FAISS similarity search with contextual prompts for better grounding.

 Challenge: LLM latency and memory usage.
Solution: Used local LLMs with Ollama for faster, offline inference.

Future Improvements:

 Add support for multiple PDFs and dynamic dataset updates.

 Implement feedback-based recommendation refinement.

 Introduce filters (e.g., genre, author) for more tailored suggestions.

 Deploy as a web app with user login and history tracking.

Conclusion:

Bookbuddy demonstrates an effective integration of vector-based retrieval and local language models to generate book recommendations. The combination of **FAISS**, **SentenceTransformers**, and **Streamlit** creates a personalized, privacy-friendly chatbot experience. It offers a solid foundation for scalable and intelligent recommendation systems.

References:

- Streamlit Documentation
- LangChain Documentation
- FAISS by Facebook AI
- SentenceTransformers
- Ollama – Run LLMs Locally

-