**Sravani Pati**
**CUID: C79908900**

**Prof: Paige Anne Rodeghero**
**CPSC 8740**

# Final Project

# Textbook Tutor: An AI-Powered Interactive Learning Assistant

**Streamlit Link : [Project Link] Github Link: [Repo Link]**

## 1. Project Overview

The goal of this project is to transform traditional textbook learning into an interactive, AI-driven experience. Building on my earlier single-book prototype, this project expands into a full platform where users can upload any textbook, explore it across multiple modes (summaries, storytelling, business cases, quizzes, and free-form Q&A), and interact with an AI tutor powered by Retrieval-Augmented Generation (RAG).

The system consists of:

- **Backend:** FastAPI RAG service deployed on Render.
- **Frontend:** Streamlit UI deployed on Streamlit Cloud.
- **AI Models:** OpenAI chat models + embeddings.

This semester's work focuses on multi-book support, a complete user interface, full backend deployment, and a polished interactive learning experience.

## 2. Project Timeline with Weekly Milestone

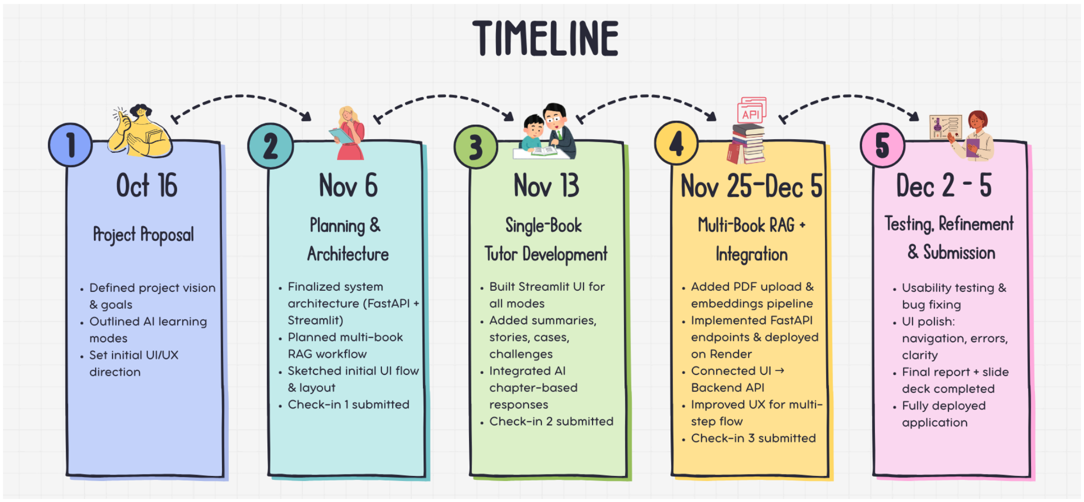
TIMELINE
1 Oct 16 Project Proposal
• Defined project vision & goals
• Outlined AI learning modes
• Set initial UI/UX direction
2 Nov 6 Planning & Architecture
• Finalized system architecture (FastAPI + Streamlit)
• Planned multi-book RAG workflow
• Sketched initial UI flow & layout
• Check-in 1 submitted
3 Nov 13 Single-Book Tutor Development
• Built Streamlit UI for all modes
• Added summaries, stories, cases, challenges
• Integrated AI chapter-based responses
• Check-in 2 submitted
4 Nov 25-Dec 5 Multi-Book RAG + Integration
• Added PDF upload & embeddings pipeline
• Implemented FastAPI endpoints & deployed on Render
• Connected UI → Backend API
• Improved UX for multi-step flow
• Check-in 3 submitted
5 Dec 2 - 5 Testing, Refinement & Submission
• Usability testing & bug fixing
• UI polish: navigation, errors, clarity
• Final report + slide deck completed
• Fully deployed application
API

**Fig: Project Timeline with weekly Milestone**

The project plan was executed across 5 distinct milestones, outlined in the timeline below, with specific deliverables for each week. This timeline highlights the step-by-step evolution of the

Textbook Tutor project. The early phase focused on defining goals and designing both the system architecture and the initial UI/UX flow. The middle stages involved rebuilding the single-book tutor in Streamlit, developing multi-book RAG capabilities, and deploying a fully functional backend. The final stage emphasized usability testing, UI improvements, error handling, and preparation of deliverables. By following this structured plan, the project achieved a fully deployed, user-friendly, AI-powered learning platform within the given timeframe.

## 3. AI Tool Setup

For this project, I selected AI tools and platforms that were simple to integrate, reliable for learning applications, and suitable for building both the **single-textbook AI tutor** and the new **multi-book RAG tutor**. This version improves significantly on my earlier Colab-based prototype by using stronger models, stable cloud deployment, and cleaner integration.

## AI Tools & Reasons for Selection

### OpenAI Models (Chat + Embeddings)

Used for question answering, summaries, stories, business cases, quizzes, and general tutoring responses. OpenAI was chosen because it provides high-quality outputs, handles tokenization well, and integrates cleanly with Python.

### Previous Models (DeepSeek + Mistral 7B)

In my earlier version, the single-textbook tutor used DeepSeek-processed chapters and retrieval with Mistral 7B in Google Colab. While it worked, it was slow, not deployable, and limited by Colab resources.

### Single-Textbook Dataset (High Quality)

The single-textbook tutor remains strong because it uses a curated dataset I created earlier: chapter summaries and **five manually refined Q&A pairs per chapter**. This gives the system very accurate and polished answers.

### RAG for Multi-Book Tutor (with Limitations)

The new multi-book tutor uses RAG (PDF extraction → chunking → embeddings → retrieval) to support any uploaded textbook. It provides grounded answers across multiple books, but its accuracy depends on the quality of the uploaded PDFs and is not as precise as the curated single-book dataset.

## Platforms and Their Appropriateness

**FastAPI (Backend)**
Lightweight, fast, and ideal for the endpoints /books, /ingest_pdf, and /ask. Handles ingestion and retrieval logic cleanly.

**Render (Backend Hosting)**
Used for deploying FastAPI with secure environment variables and a public HTTPS URL that the frontend can call reliably.

**Streamlit (Frontend UI)**
Chosen for rapid, code-friendly UI development. Supports file uploads, learning modes, and real-time responses without traditional web development.

## Development Environment & Libraries

- Python 3 virtual environment
- Libraries: fastapi, uvicorn, streamlit, requests, PyPDF2, OpenAI Python SDK
- All dependencies tracked in requirements.txt
- Secrets stored using Render Environment Variables & Streamlit Cloud Secrets
- Backend tested locally with uvicorn before deployment.

## Setup Challenges & Resolutions

I faced several challenges while setting up the system. The first issue involved incorrect API keys, which caused repeated 401 errors until I generated a new key and updated the secrets in both Render and Streamlit Cloud. Although the project worked perfectly on my local machine, deployment introduced new difficulties because Streamlit Cloud cannot access localhost, which caused the backend to fail until I replaced the local address with the public Render API URL. Some functions and imports also behaved differently once deployed, so I had to adjust and rewrite parts of the backend and frontend to ensure they worked consistently. At one point the backend and UI were not communicating at all, which led me to separate the project into two fully independent deployments, one for the FastAPI backend on Render and one for the Streamlit interface on Streamlit Cloud. I also had to refactor my earlier Colab-based code, since it was not designed for deployment, and rebuild the logic to fit Streamlit's structure. Despite these issues, the final setup is now stable and supports both the refined single-textbook tutor and the new multi-book RAG tutor.

## 4. User Interface design and Integration

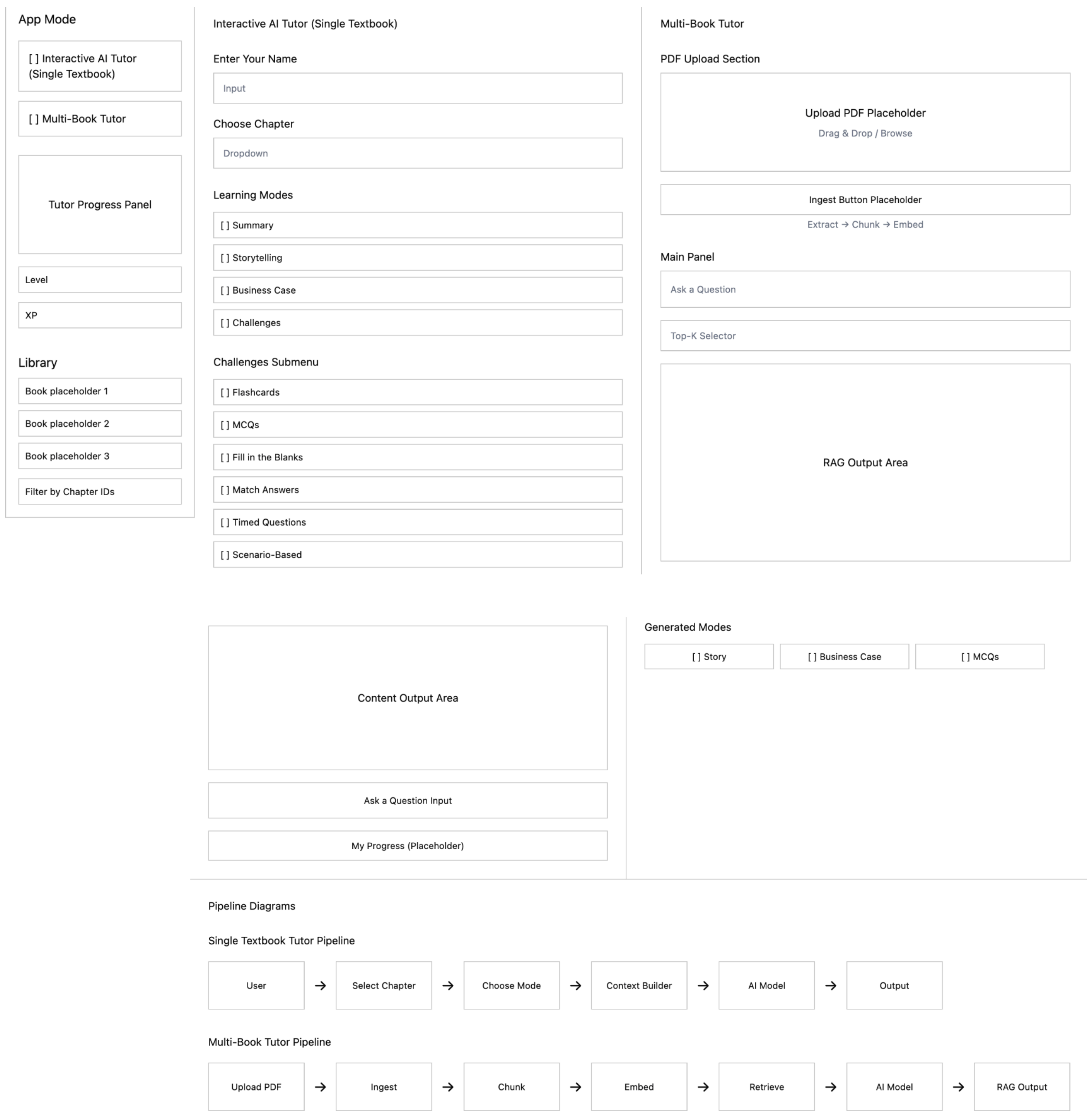

**Fig: Low fidelity framework for Interactive AI Tutor**

**Link to access : [Low fidelity framework for Interactive AI Tutor]**

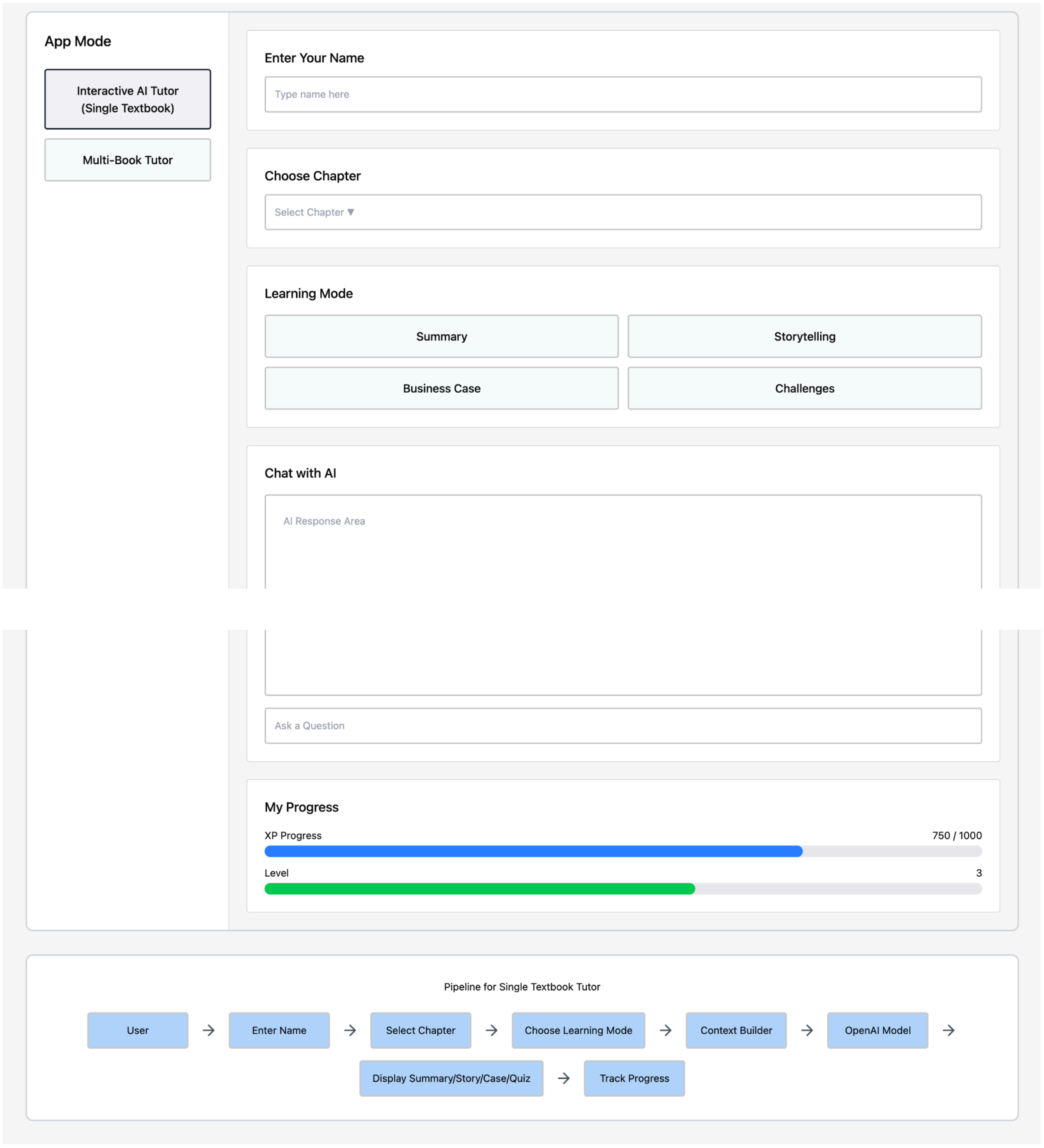

**Fig: High-Fidelity Wireframe of the Interactive AI Tutor (Single Textbook) and its Pipeline**

**Link to access:** **Interactive AI Tutor single text book High Fidelity Wire Frame and Pipeline**

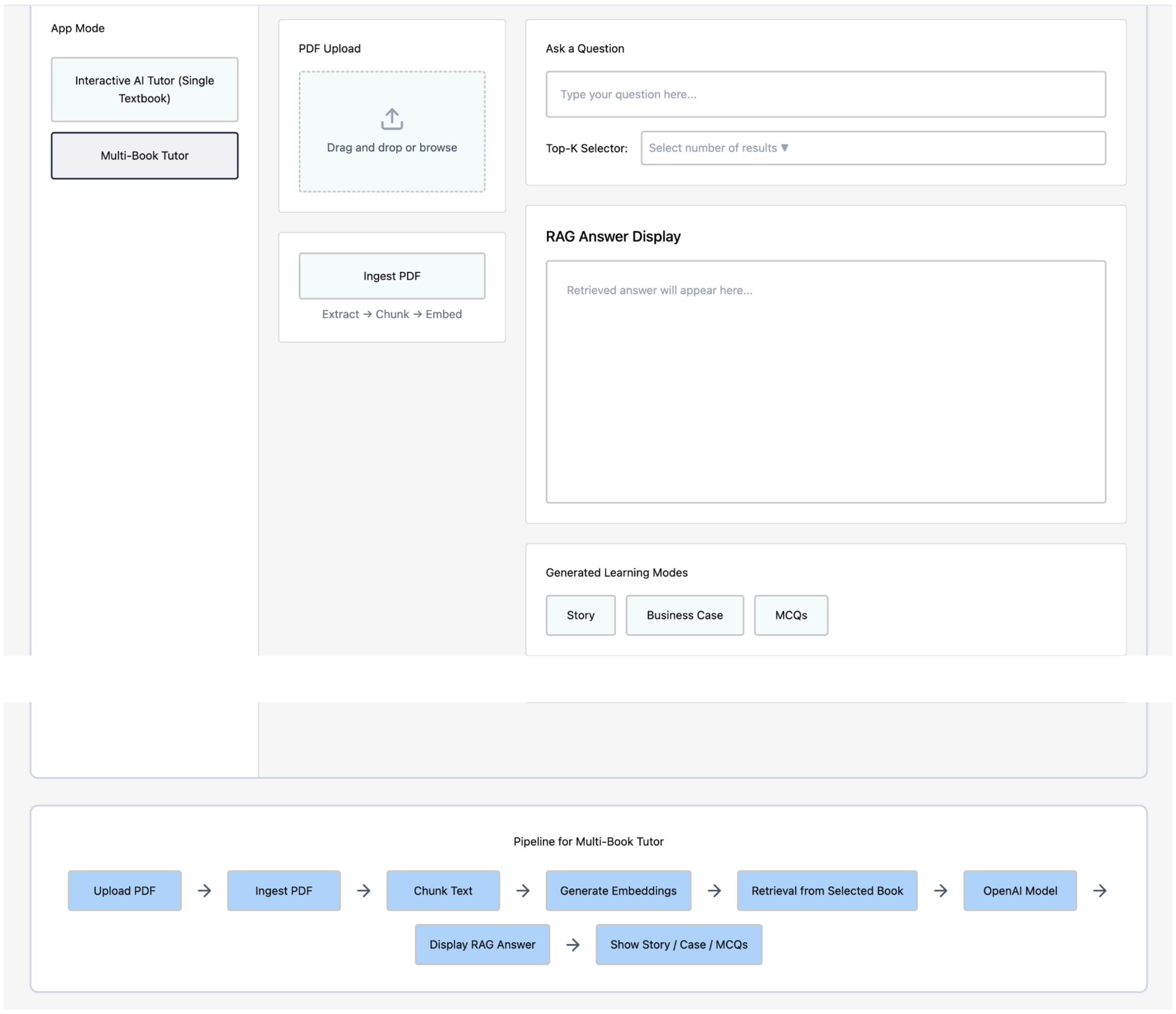

**Fig: High Fidelity Wireframe of Multi-book Tutor**

**Link to access:** **High Fidelity Wireframe of Multi-book Tutor**

I approached the user interface design in a practical and efficient way because the main focus of this project is the AI functionality and the multi-book retrieval system. I did not want to spend unnecessary time manually drawing screens when the goal was to make the learning experience strong and functional. I used ChatGPT to refine my layout ideas and describe exactly how the screens should look and behave. After shaping those ideas through prompts, I used Figma Make, the free AI feature in Figma, to instantly generate the low-fidelity and high-fidelity wireframes. Whenever the layout did not match what I had envisioned, I rewrote the prompt, regenerated options, and made adjustments until the interface fully matched the flow I had imagined for the final website. This approach helped me move quickly and still create thoughtful, organized wireframes that clearly represent the structure of the application.

After finalizing the wireframes through Figma Make, I used them as a guide to build the real interface in Streamlit. Streamlit was ideal because it allows fast and clean UI development while keeping everything simple and accessible for learners. I recreated the sidebar, navigation flow, input fields, chapter selectors, and content panels exactly as designed in the wireframes. The interface is minimal, clean, and organized so that users always know where to look and how to interact with the system.

The UI connects directly to the FastAPI backend that I deployed on Render. Every user action, such as selecting a chapter, choosing a learning mode, uploading a PDF, or asking a question, communicates with backend endpoints that handle ingestion, chunking, embeddings, and retrieval. Streamlit's session state features made these interactions smooth because they allowed the content to update instantly without refreshing the entire app.

I kept the sidebar constant for both the single-textbook tutor and the multi-book tutor so that the experience feels familiar and consistent. Only the main content area changes based on the user's selection. This design choice follows the wireframes closely and creates an interface that is intuitive even for first-time users.

During integration, I faced a few challenges. The biggest issue was that Streamlit Cloud cannot access localhost, so I had to replace all local backend links with the publicly deployed URL from Render. I also had to rebuild many UI portions from the older Colab version because Streamlit requires a different approach for structuring code and handling state. Through repeated testing, I refined the layout, improved spacing, and made the flow clearer so that users can navigate effortlessly.

The final interface is simple, functional, and easy to use. It matches the wireframes closely and supports real-time interactive learning. The layout is clear, the learning modes are easy to access, and the main content panel updates instantly. The combination of thoughtful wireframing, careful design decisions, and smooth integration created a user interface that enhances the experience of studying through an AI-powered textbook tutor.

## 5. Testing and Refinement

My testing and refinement process built on my earlier Google Colab version of the Interactive AI Tutor, which had already been evaluated in a mixed-methods user study with sixty two participants. That study showed high usability and engagement and helped me understand which features learners found most helpful, such as multiple learning modes, guided walkthroughs, and progress tracking. These results motivated me to rebuild the system as a cleaner, easier web app using Streamlit and FastAPI.

For this deployed version, I did informal usability testing with about five to eight friends and roommates who had used the Colab notebook before. I asked them to try both the single textbook tutor and the multi book tutor, upload a PDF, ingest it, ask questions, and explore the learning modes. They consistently said that the new web interface felt much easier and more comfortable than working inside Colab, especially because everything is organized in the sidebar and main content area instead of code cells.

Based on their feedback, I improved labels and helper text around the upload and ingest steps, added clearer status and error messages when the backend or API key fails, and adjusted layout spacing so that main actions are easier to see. Some limitations remain, including occasional delays when the Render backend wakes up and the fact that the multi book RAG pipeline is still less mature than the curated single textbook tutor. A more formal usability study for this new deployment is planned as future work, but current testing already shows a clear improvement in usability over the original Colab version.

## 6. Conclusion

This project transformed my earlier Colab-based prototype into a fully deployed, easy-to-use web application that supports both a curated single-textbook tutor and a flexible multi-book RAG system. By combining FastAPI, Streamlit Cloud, and OpenAI's models, I created a learning platform that is simple, stable, and accessible for students. The new interface is cleaner and more organized than the previous version, and informal testing already shows a clear improvement in usability, comfort, and navigation. The wireframes guided the UI development closely, and the backend integration supports real-time responses across summaries, storytelling, business cases, challenges, and question answering. Although the multi-book RAG pipeline can still be improved, this project establishes a solid foundation for future versions with better retrieval, richer assessments, and more formal user studies. Overall, the work demonstrates a complete and functional AI-powered learning assistant that modernizes textbook learning and creates a more interactive and engaging study experience.

### 7. Future Work

In future work, I want to improve the multi-book tutor since it still has some limitations. The RAG accuracy depends on PDF quality, and retrieval is not as precise as the curated single-textbook dataset. I also faced limits with the OpenAI API because I am using a trial plan, which restricts tokens and speed; upgrading would make the system much faster and more consistent. I plan to refine chunking, improve embeddings, add better filtering, and introduce user accounts, saved progress, and multimedia explanations. A more formal usability study will also help shape the next version into a stronger and more reliable learning platform.