This is the chat link : https://chatgpt.com/share/68c9d103-09c0-8013-99c3-50d60fbdaa76

Not all prompts are included in this chat, as I also used VS Code AI chat interactions to work directly on the repository. These interactions are not shareable via link, so they cannot be included here. However, I will document and mention the prompts used. Below are the prompts used by me in this project

1.Design a scalable multi-agent architecture where a planner, researcher, and executor agent collaborate to automate a study plan generation pipeline. The agents should communicate asynchronously and handle context passing efficiently.

2.i actually want to make StudyBuddy — an AI Study Planner & Executor for students.

3.How can I integrate RAG with FAISS indexing for retrieving relevant documents efficiently in a FastAPI-based system?"

4.Generate FastAPI endpoints for creating, retrieving, and executing study plans. Include error handling, logging, and OpenAPI documentation

5.Write Python code for an executor agent that creates study guides, flashcards, and quizzes dynamically using a Gemini LLM API.

6.Optimize API response times for bulk execution of study steps and include live status tracking.

7.Write a prompt template that guides the LLM to generate step-by-step study plans based on a user's learning goal, ensuring each step has a tool assigned (RAG, Flashcards, Quiz, etc.).

8.How do I implement a fallback mechanism if Gemini API rate limits are hit, so the system retries with exponential backoff?(this was actually a very major problem)

9.After a user clicks 'Execute Step,' automatically show the result on the same page without requiring a separate 'View Result' click

10.Add a button that allows users to download their study plan as a professional-looking PDF using ReportLab

11.Add debug logs for each stage of the multi-agent workflow for easier troubleshooting during a hackathon demo.

12.How do I build a Streamlit UI that shows plan steps, executes them, and displays results dynamically?

13.What final touches can make my project hackathon-ready?

14.yes required

15.

How do I handle database migrations and ensure data integrity when adding new columns to existing tables in SQLite?

16.Create indexes on my SQLite tables to optimize queries for plan retrieval and step status updates.

17.Write code for context summarization in the researcher agent that condenses RAG search results into actionable study insights.

18 Create a validation system that ensures each generated study step has required fields like id, title, description, and tool assignment.
19.How do I implement agent coordination where the researcher agent only runs for specific tool types like RAG and FLASHCARDS?

20.Design prompts that make the LLM generate flashcards with proper question-answer pairs and difficulty levels.

21.Write code to validate LLM-generated quiz questions have correct answer formats and explanations.

22.How do I create a modern Streamlit interface with custom CSS for step cards that change color based on execution status?

23. Implement session state management in Streamlit to persist results across page interactions and button clicks.
24. Add real-time progress tracking in Streamlit that updates automatically during bulk step execution.

25. Write comprehensive API tests for all FastAPI endpoints including edge cases and error scenarios.

26.Implement connection pooling for SQLite to handle multiple concurrent API requests efficiently.

From here onwards these are some prompts in vs code AI chat

27. I think UI is still not appealing ,so please improve the UI

28. Create automated tests that verify the entire workflow from plan creation to step execution works correctly.

29. How do I package my application for easy deployment and demonstration to potential users or employers?

30. Add analytics and usage tracking to understand how users interact with the study planning system.

31. Create example study plans for different domains (programming, mathematics, science) to demonstrate versatility.

The use of AI obviously reduces the pressure of writing large amounts of code. However, there were many parts of the code provided by these LLMs that I still had to modify. Additionally, most of the comments inside the files were created with the help of LLMs to provide clear and refined explanations, as well-commented code makes the implementation more understandable.