Section 1: Title, Problem, and Use Case

Project Title: AI-Powered Interview Question Generator & Dynamic Chat Assistant

Problem & Use Case

- **Problem Statement:** Many students and job candidates struggle to find interview questions that are **tailored to their skill level and topic of interest**. Generic question banks are often too broad, lack context, and do not provide interactive guidance or hints for learning.
- **User-Centric Thinking:** The target users are students, fresh graduates, and professionals preparing for technical interviews. Their need is for a personalized, interactive environment where they can generate practice questions on specific topics, test their knowledge, and receive context-aware hints and explanations.
- **Agent's Role:** The agent’s primary function is to **generate topic-specific interview questions**, **maintain session-based context**, and **provide dynamic answers or hints** based on user queries. It serves as a virtual interview coach that adapts to the user’s learning progress.
-

Section 2: Agent Description

- **Functionality:** The AI agent enables users to generate interview questions for any topic and difficulty level, interactively ask for hints or answers, and maintain multiple independent chat sessions for organized practice.
- **Key Components:**
    1. Question Generation: Produces structured interview questions using user-specified topics and difficulty levels.

    2. Context-Aware Chat: Maintains chat history for multi-turn interactions and session-based context.

    3. Answer / Hint Module: Responds to user queries with explanations, hints, or full answers.

4. Project Documentation Generator: Summarizes session conversations and outputs a structured project report..

Section 3: Final Prompt

Prompt 1: Formatting Output

➢ I want to format the output in a way that always gives users a consistent result

- **Rationale:** This prompt is designed to ensure that the output generated by the model is structured and predictable, regardless of variations in user input. By emphasizing consistent formatting, it reduces ambiguity and makes it easier for downstream applications (e.g., dashboards, reports, or further processing) to parse and use the generated data. This approach also enhances user experience by creating uniformity, which helps users quickly understand and act on the outputs.

Prompt 2: The Application becomes slow after deploying in Streamlit Cloud

➢ this is taking too much time to load in cloud: # RunnableSequence agent_sequence = ({ "topic": lambda x: x["topic"], "number": lambda x: x["number"], "level": lambda x: x["level"] } | input_prompt | model | parser | (lambda x: plan_prompt.format(topic=x.topic, number=x.number, level=x.level)) | model | (lambda x: output_prompt.format(topic=topic, number=number, level=level, planned_questions=x.text)) | model | final_parser)

- **Rationale:** This prompt highlights a performance issue caused by the sequential execution of multiple model calls and lambda transformations in the `RunnableSequence`. Each stage in the pipeline invokes the model separately, which can significantly increase latency, especially when deployed in cloud environments with limited resources or network delays. The prompt is designed this way to illustrate a complex, multi-step question generation pipeline and to prompt consideration of optimization strategies like batching, caching, or reducing redundant model calls to improve response times.

Prompt 3: Chatting Feature After Question Generation

➢ Add a chat feature so that users can continue chatting after the question generation, like: give answers to these questions", make it a dynamic chatbot

- **Rationale:** This prompt aims to transform the static question-generation process into an interactive experience, allowing users to continue the conversation dynamically. By integrating a chat feature, the system can maintain context, respond to follow-up questions, and handle iterative inputs, enhancing user engagement. The design ensures that the model can act both as a question generator and as a conversational agent, supporting a seamless transition from content generation to interactive guidance.

Prompt 4: Separate Sessions and New Chat Feature

➢ It is continuing in the same chat. How to separate sessions, or add a new chat feature

- **Rationale:** This prompt addresses session management in a conversational AI application. By enabling separate sessions or a "new chat" feature, each interaction can start with a clean context, preventing previous conversation history from interfering with the current dialogue. This design ensures privacy, context isolation, and a better user experience by allowing multiple independent conversations. It is particularly useful in applications where users may want to explore different topics or scenarios without residual context affecting responses.

Prompt 5: User Interface Updated

➢ Provide a link in home page itself to start a chat (like all chatbot - chatgpt, gemini provides along with new chat feature and update the topic, number and difficulty level and generate button in the home section and new chat only in sidebar (for better fiendly environment). While switching to new session clean previuos session input

- **Rationale:**
    1. **Provide a link/button on the home page to start chat (like ChatGPT, Gemini):**
        - Users expect a **clear entry point** to begin.
        - It feels natural and familiar since popular chatbots work this way.
        - Reduces confusion—new users don’t need to search the sidebar first.
    2. **Move topic/number/difficulty inputs & “Generate” button to the main page (home section):**
        - Keeps the **main interaction front and center** (where users focus most).
        - Sidebar stays clean for navigation (just switching sessions).
        - Makes the UI more user-friendly and less cluttered.
    3. **Keep only “New Chat” in sidebar:**

- Sidebar works best for **navigation between sessions**.
- Prevents mixing controls (inputs vs navigation) → avoids confusion.

4. **Clear previous inputs when switching sessions:**
    - Prevents old values from “leaking” into the new chat.
    - Each session feels **fresh and independent**, just like a new conversation in ChatGPT.
    - Improves usability and avoids accidental mistakes (e.g., generating ML questions when you wanted Python).

Section 4: Exploration Log (Process & Iteration)

Attempt 1:

- **Initial Approach:**
  The first implementation used a single LLM call to generate interview questions without maintaining any session history. The model was asked to generate questions, and answers or hints were provided in isolated calls without context.
- **Why didn’t it work?**
  Multi-turn interactions became incoherent because the AI had no memory of previous questions or user queries. Follow-up hints or clarifications were inconsistent, and users could not continue a meaningful conversation about the generated questions.

Change:

- Introduced **session-based chat history** to track multi-turn interactions.
- Implemented **structured JSON output** for question generation to ensure consistent formatting and easy parsing.
- Separated prompt responsibilities into **modular components**: question generation, answering questions, session context retrieval, and documentation generation.

Attempt 2:

- **New Approach:**
  Each session now maintains its own `chat_history`, storing both generated questions

and user interactions. Prompts were refined to include context from previous messages when answering follow-up queries. A dedicated prompt was also added to generate the project documentation automatically from the session history.

- **Result:**
  Chat became coherent and context-aware. Users could ask questions about previously generated questions, receive hints, and dynamically interact with the AI in a way that simulates a real interview environment. The documentation prompt successfully summarized the session history into structured reports.

Attempt 3 (Optional Enhancements):

- **Approach:** Added a **“New Chat” feature** to allow multiple independent practice sessions and avoid context contamination between different topics or users.

- **Result:** Users could now maintain multiple interview practice threads simultaneously, increasing usability and flexibility.

Key Learnings:

1. **Context preservation is critical** for multi-turn interactive systems.

2. **Structured outputs (JSON)** simplify parsing, integration, and reuse across modules.

3. **Modular prompt design** reduces overlap and improves maintainability.

4. Iterative testing and refinement are necessary to handle edge cases, such as empty inputs or ambiguous user queries.

5. Adding session management and project documentation generation greatly enhances the usability and completeness of the system.

Section 5: Output Examples

Provide a clear demonstration of your agent's behavior. Show examples of both good and bad inputs and how the agent responds.

- **Good Input Example:**
  - **Input:** Topic: gen ai, Number: 5, Difficulty: Medium
  - **Output:**
  - Q1. Explain the core architectural differences and advantages of transformer models compared to earlier recurrent neural networks (RNNs) or convolutional

neural networks (CNNs) in the context of generative AI applications like large language models.

  - Q2. When developing a Gen AI application, what are some key considerations for effective prompt engineering, and how would you approach iteratively optimizing prompts for a specific business use case, such as content generation or customer service automation?
  -
  - Q3. Beyond basic bias detection, what are some more nuanced ethical challenges or potential misuses of generative AI models in real-world deployment, and what strategies would you propose to mitigate these risks?
  -
  - Q4. How would you evaluate the performance and quality of a generative AI model for a task where traditional quantitative metrics (e.g., accuracy, precision) might be insufficient, such as creative writing, image generation, or code synthesis? What qualitative and quantitative approaches would you combine?
  -
  - Q5. Discuss the concept of 'model hallucination' in large language models. What causes it, what are its implications for practical applications, and what current research or engineering strategies are being explored to reduce its occurrence?
  - **Analysis:** This output is successful because:

    1. **Relevance**: Questions directly relate to the specified topic ("Gen AI") and are appropriate for the selected difficulty level (Medium).

    2. **Depth**: Each question requires critical thinking and application of knowledge, which is suitable for interview preparation.

    3. **Clarity**: Questions are well-structured, detailed, and unambiguous, making them easy for users to understand and attempt.

    4. **Diversity**: The set covers technical, ethical, and evaluation aspects of Gen AI, ensuring comprehensive coverage.

- **Bad Input Example:**
  - **Input:** ""
  - **Output:** ⚠️ Please enter a valid topic to generate questions.
  - **Analysis:** This output is not fully successful because:

    1. **No questions are generated:** The AI cannot proceed with a blank topic, so the user does not get any content.

    2. **Improvement opportunities:**

- Implement **input validation** in the UI to prevent empty submissions.
- Provide **suggested topics** or examples when the field is blank, so users can continue without manual correction.
- Include **clarifying questions** in the AI response, e.g., “Do you want me to suggest popular topics like Machine Learning, NLP, or Computer Vision?”

Section 6: Reflections

Challenges:

1. **Maintaining coherent multi-turn conversations:** Initially, the AI lost context between follow-up queries, making it difficult to provide consistent answers or hints.

2. **Handling edge cases and invalid inputs:** Blank topics, negative question numbers, or ambiguous user queries caused either silent failures or inconsistent AI responses.

3. **Balancing depth and clarity in prompts:** Designing prompts that generated technically accurate questions while remaining clear and understandable required multiple iterations.

4. **Integrating session-based multi-chat support:** Implementing independent chat sessions with persistent history added complexity to state management in Streamlit.

Learnings:

1. **Importance of session context:** Preserving chat history is crucial for multi-turn AI interactions, particularly when the user refers back to previously generated content.

2. **Modular prompt design improves maintainability:** Separating question generation, answer/hint generation, context retrieval, and documentation generation into distinct prompts made the system more robust and easier to debug.

3. **Structured output is essential:** Using JSON outputs ensured consistent parsing and integration across different modules, reducing errors and simplifying downstream tasks.

4. **Iterative refinement enhances user experience:** Frequent testing, analyzing outputs, and adjusting prompts incrementally helped achieve high-quality, user-centered

behavior.

Future Work:

1. **Adaptive difficulty levels:** Implement logic to adjust question difficulty dynamically based on the user's previous answers.

2. **Expanded topic coverage:** Support additional domains beyond technical interviews, such as behavioral questions or domain-specific case studies.

3. **Automated evaluation and feedback:** Add features for the AI to assess user answers and provide detailed scoring or hints.

4. **Topic suggestions and smart input handling:** Suggest popular topics automatically when users leave the topic blank, improving usability.

5. **Enhanced multi-session management:** Allow users to save, export, and resume sessions, creating a more complete study tool.