Conversational AI Assistant for URI Institutional Review Board (IRB)

Suparna Veeturi, Justin Watkins, Resit Sendag

GitHub Repository: <https://github.com/veeturisuparna/URI-IRB-AI_Assistant>

# 1. Introduction & Problem Statement

Preparing Institutional Review Board (IRB) submissions at the University of Rhode Island (URI) is a critical but often complex process for researchers. It involves extensive paperwork, including the main Application, Consent forms, and various supporting documents, such as Appendix E for studies involving medical devices. Researchers must adhere to specific language, formatting guidelines, and institutional policies, which can be time-consuming and challenging, especially when managing multiple projects. Errors or inconsistencies can lead to delays in research approval.

The overall goal of this project was to explore the potential of generative AI to alleviate these challenges by creating a conversational AI assistant designed to streamline the preparation of IRB submission packages. The vision included leveraging AI to suggest standard language based on templates and past examples, guide users through required sections, and help manage documentation.

As a first step towards this larger goal, and to create a feasible Minimum Viable Product (MVP) within the scope of EGR 404, this project focused specifically on streamlining the creation of Appendix E for medical device studies. Appendix E has a defined structure but requires detailed information about the device and its use within the specific research context. The implemented MVP, the "Guided Appendix E Generator," aims to assist researchers by guiding them through the required fields for this form via an interactive chat interface, leveraging AI for conversation management and potentially for information retrieval.

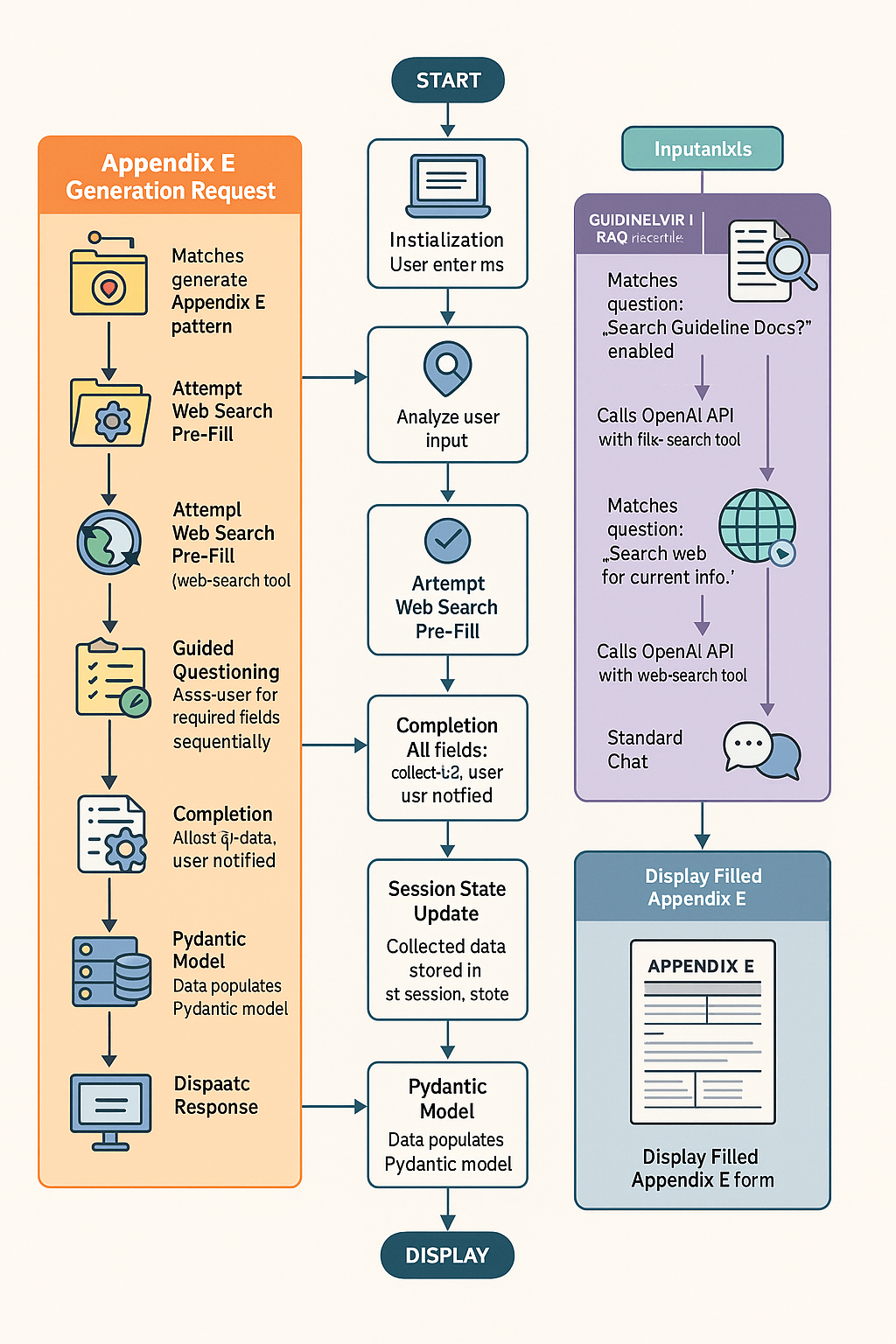
Okay, here is the revised Methodology & Implementation section incorporating the RAG functionality:

## **2. Methodology & Implementation**

**Project Overall Flow Description:**

The application functions as an interactive chatbot within a Streamlit web interface. The core workflow proceeds as follows:

1. **Initialization:** The user starts the Streamlit application (app.py). The interface loads, displaying the chat history (initially empty or with a welcome message) and sidebar options (like enabling RAG/Vector Store search or Web Search).
2. **User Input:** The user types a message into the chat input box.
3. **Input Analysis:** The application analyzes the user's input to determine the intended action:
   * **Appendix E Generation Request:** If the input matches a pattern like "Generate Appendix E for [Device Name]", the application extracts the device name and enters the "Appendix E Mode".
     + **Web Search Pre-fill (Attempt):** It first calls the OpenAI API with a specialized prompt, instructing it to use the web\_search tool to find publicly available specifications for the named device and determine the likely section (A or B). This attempts to pre-populate some fields in the background.
     + **Guided Questioning:** The application then begins a multi-turn conversation, asking the user for required Appendix E fields that are still missing (starting with study-specific ones like PI, Title, Proposed Use). User answers are stored in Streamlit's session state, populating a Pydantic model.
     + **Completion:** Once all required fields are collected, the user is notified. The collected data is available in the application state (and potentially displayed in a debug view).
   * **Guideline/RAG Question:** If the input is a general question AND the "Search Guideline Docs?" (RAG) option is enabled in the sidebar:
     + The application calls the OpenAI API, including the user's question and the file\_search tool configured with the specific Vector Store ID.
     + The API retrieves relevant context from the Vector Store and uses it along with the LLM's knowledge to generate a synthesized answer.
   * **Web Search Question:** If the input is a general question AND the "Search web for current info?" option is enabled (and RAG is not):
     + The application calls the OpenAI API, including the user's question and the web\_search tool.
     + The API uses web search results to formulate an answer.
   * **Standard Chat:** If none of the above conditions are met, the input is treated as a standard chat message:
     + The application calls the OpenAI API with just the user's message.
     + The API generates a response based on the LLM's general knowledge.
4. **Display Response:** The application displays the AI's response (whether it's the next question in Appendix E mode, a RAG/Web answer, or a standard chat reply) in the chat interface.



*Overall Flow for IRB document generation*

1. **Loop:** The application waits for the next user input, maintaining the conversation state (including collected Appendix E data) via st.session\_state.

The project resulted in a multi-functional Minimum Viable Product (MVP) developed as a web application using Python and the Streamlit framework. It leverages the OpenAI API (gpt-4o-mini) for its core generative AI capabilities, integrating conversational AI, tool use (web search, file search), and Retrieval-Augmented Generation (RAG).

### **2.1. Tools and Technologies Used**

* **Programming Language:** Python 3.12,5
* **Web Framework/UI:** Streamlit
* **AI Model:** OpenAI API (gpt-4o-mini)
* **Data Modeling:** Pydantic (for defining the structure of Appendix E Sections A and B)
* **Key Libraries:** openai, streamlit, python-dotenv, pydantic
* **Vector Store:** OpenAI Vector Store service (for RAG)
* **Development Environment:** Visual Studio Code, Python Virtual Environments (venv)
* **Version Control:** Git, GitHub

### **2.2. Core Functionalities**

The MVP provides three primary modes of interaction accessible through the Streamlit interface:

**Mode 1: Guided Appendix E Generation**

This is the core workflow for generating the Appendix E document:

1. **Initiation & Device Identification:** The user initiates the process by providing the name of the medical device in the chat.
2. **Automated Pre-fill Attempt (Experimental):**
   * The application triggers an LLM call using a specific prompt instructing it to use the web\_search tool.
   * The goal is to automatically determine the likely Appendix E section (A or B) and extract publicly available device specifications (Manufacturer, Regulatory Status, Classification, Description, Adverse Effects) into a JSON format.
   * The prompt explicitly excludes study-specific fields.
   * The extracted data attempts to pre-populate a Pydantic model (AppendixESectionA or AppendixESectionB). *(Note: Add your assessment of this feature's reliability here).*
3. **Interactive Data Collection:**
   * The application then enters a turn-by-turn conversational flow managed using st.session\_state.
   * It systematically prompts the user for any required fields remaining empty in the Pydantic model, prioritizing study-specific details (Principal Investigator, Project Title, Proposed Use, Rationale, Risk Determination, etc.).
   * Enum fields present choices, and input is validated.
   * User responses update the Pydantic model instance stored in the session state.
4. **Completion & Output:** Once all necessary fields are populated, the user is notified.

The primary output is the structured data collected within the Pydantic model, visible in the application's debug/sidebar view.

**Mode 2: Retrieval-Augmented Generation (RAG) for Guideline Q&A**

This mode allows users to ask general questions about URI IRB guidelines or potentially query uploaded examples:

1. **Vector Store Setup:** An OpenAI Vector Store was utilized, intended to be populated (manually via OpenAI tools/API outside the app) with relevant documents like the URI IRB guidelines and potentially past Appendix E examples. The specific Vector Store ID (vs\_6812a49209d0819188df293cea4867f1) was hardcoded in the application.
2. **User Query:** The user asks a question in the chat interface.
3. **Tool Activation:** If the "Search Guideline Docs?" checkbox (or equivalent logic triggering file\_search) is enabled in the UI, the application includes the file\_search tool in its API call to OpenAI, targeting the specified Vector Store ID.
4. **Retrieval & Generation:** The OpenAI backend performs a semantic search over the indexed documents in the Vector Store to find relevant context based on the user's query. This retrieved context is then provided to the LLM (gpt-4o-mini) along with the original query.
5. **Synthesized Answer:** The LLM generates an answer based *both* on the retrieved context from the documents and its general knowledge, providing a more informed and specific response than it could alone.

**Mode 3: General Chat / Web Search Q&A**

The application can also function as a general chatbot:

1. **Standard Chat:** If the input is not recognized as an Appendix E initiation and RAG is not enabled, it functions as a standard chatbot using the LLM.
2. **Web Search Integration:** If the "Search web for current info?" checkbox is enabled, the web\_search tool is included in the API call, allowing the chatbot to answer questions requiring up-to-date information from the internet.

**Data Modeling with Pydantic:**

Pydantic models (models.py) remained crucial for the Guided Appendix E Generation mode, defining the structure for Sections A and B, enabling data validation, and providing metadata for generating user prompts.

### **2.3. Generative AI Integration**

Generative AI (gpt-4o-mini) was integrated across the functionalities:

* **Natural Language Understanding:** Parsing user input to determine intent (start Appendix E, ask guideline question, general chat).
* **Tool Use:** Orchestrating calls to web\_search (for pre-fill and general Q&A) and file\_search (for RAG Q&A) via the OpenAI API.
* **Data Extraction:** Attempting structured JSON extraction from web results during the pre-fill phase.
* **Conversation Management:** Driving the turn-by-turn Appendix E data collection and generating appropriate prompts.
* **Retrieval-Augmented Generation:** Synthesizing answers by combining user queries with context retrieved from the Vector Store.
* **Standard Generation:** Providing answers based on its internal knowledge or web search results for general queries.

Okay, here is Section 4: Design Choices & Challenges:

## **4. Design Choices & Challenges**

Several key design decisions and challenges shaped the development of the MVP:

**Design Choices:**

* **Streamlit UI:** Chosen for its simplicity in creating interactive chat applications and managing session state, which was essential for the multi-turn conversational flow.
* **Pydantic Data Modeling:** Adopted to enforce a strict structure for the Appendix E data, facilitate validation, and easily access field metadata (like descriptions) for generating prompts.
* **Multi-Turn Conversation for User Input:** Decided to prioritize gathering essential user-specific information (PI, Title, Proposed Use, Rationale) through direct questions *before* attempting automated generation or lookup for remaining fields. This ensures critical context is captured accurately.
* **Integration of Multiple Modes:** Designed the application to handle Appendix E generation, RAG-based guideline Q&A, and general web search Q&A within a single interface, using sidebar toggles and input analysis to switch contexts.
* **Web Search Pre-fill (Experimental):** Included as an attempt to reduce manual data entry for publicly available specs, recognizing it as a complex task reliant on LLM data extraction capabilities.

**Challenges:**

**Orchestrating Multi-Source Information Gathering:** A primary challenge was designing the assistant to gather information in a specific, logical **hierarchy** rather than querying all sources indiscriminately. The desired **source prioritization** was:

1) Explicit **user input** for study-specific details,

2) Context from the **Vector Store** (guidelines/examples via RAG and file\_search)

3) General information from the **Web Search** (web\_search) for remaining device specifications.

**Field-Specific Sourcing & Conditional Logic:** Implementing this hierarchy required complex **conditional logic** within the application (app.py). The system needed to differentiate between field types:

* Fields *always* require user input (e.g., Principal Investigator, Proposed Use).
* Fields *potentially* found in the Vector Store (e.g., standard phrasing, template structures).
* Fields *likely* needing web search (e.g., manufacturer details, current regulatory status, common adverse effects). This required careful **state management** to track which fields were collected and sophisticated **prompt engineering** to guide the LLM during tasks like the pre-fill attempt.

**Tool Orchestration:** Managing the sequence—first complete the user-input phase, then potentially trigger file\_search or web\_search for *specific remaining fields*—was a significant **tool orchestration** challenge. It demanded precise control over when and how API calls were made, ensuring the LLM respected the **information hierarchy** established by the application logic, rather than simply defaulting to its general knowledge or the most accessible tool. This wasn't about "training" the model in the fine-tuning sense, but rather about controlling its execution flow through careful prompting and application-level logic.

## **5. Innovation and Practical Impact**

This project innovatively applies generative AI by creating a **targeted assistant for the specific URI IRB Appendix E form**, integrating **conversational guidance**, **Retrieval-Augmented Generation (RAG)** for institutional context, and **web search** for device specs using a **hierarchical information strategy**.

The practical impact for URI researchers includes:

* **Reduced Time & Effort:** Streamlines the tedious form-filling process.
* **Improved Accuracy & Compliance:** Helps ensure all required fields are addressed consistently.
* **Increased Accessibility:** Lowers the barrier for completing IRB paperwork, especially for new researchers.
* **On-Demand Guideline Access:** Provides quick answers to policy questions via RAG.

Ultimately, this MVP serves as a valuable proof-of-concept demonstrating AI's potential to simplify essential administrative tasks in research.

## **6. Milestones and Timeline**

| **Tasks** | **W1** | **W2** | **W3** | **W4** |
| --- | --- | --- | --- | --- |
| Milestone 1: Project Setup & Foundational Code Complete (Continuation from Lab 6) |  |  |  |  |
| Milestone 2: Core Conversational Logic Implemented |  |  |  |  |
| Milestone 3: System Integration & MVP Testing Completed |  |  |  |  |
| Milestone 4: Experimental RAG / Automated Lookup Feature |  |  |  |  |
| Milestone 5: Additional testing and submissions |  |  |  |  |

## **7. Conclusion and Future Work**

This project successfully developed a functional MVP of a conversational AI assistant tailored for the URI IRB Appendix E form. By integrating guided data collection, RAG for guideline context, and web search capabilities within a Streamlit interface, the tool demonstrates a practical approach to simplifying a key administrative task for researchers, saving time and potentially improving compliance.

**Future Work:**

The core architecture and methodologies employed in this MVP provide a strong foundation for significant expansion. The most logical next step is to **scale the assistant to handle the complete URI IRB submission package**, including the main Application form and Consent documents. This would involve:

* Modeling the structure of these additional forms (likely using Pydantic).
* Extending the conversational logic to guide users through all required sections of the full submission.
* Potentially enhancing the RAG capabilities with a broader set of institutional templates and anonymized examples relevant to all forms.
* Implementing robust PDF generation for the entire package.

Further enhancements could include separate chat managements for different projects.