Project Report: Medical Copilot Development

1. Introduction

The goal of the Medical Copilot project was to develop a simple, private, and efficient Streamlit application to assist medical professionals. The system was designed to leverage local Large Language Models (LLMs) to analyze patient records, provide diagnostic suggestions based on a custom medical knowledge base, and handle both text and image data securely on a local machine. Throughout the development process, several key challenges were encountered and overcome.

---

2. Challenge: Handling Diverse Patient Data (Text and Images)

- **The Challenge**: A primary requirement was for the application to process both textual medical reports (PDFs) and visual data like X-rays and MRIs. The initial idea was to automatically extract images embedded within the PDF reports. This proved to be technically complex and unreliable, as image quality could degrade and context could be lost.
- **The Solution**: We simplified the workflow to improve robustness and user control. Instead of automatic extraction, we implemented a dual-input system:
    1. **Patient Onboarding**: Doctors can upload a patient's PDF report and their associated medical images as separate files.
    2. **Chat Interface**: The UI was designed with a dedicated image uploader within the chat column. This allows the doctor to explicitly select which image they want to analyze alongside their text-based query. This approach keeps the text (RAG) and vision analysis pipelines separate and ensures the model is always analyzing the correct, user-selected image.

---

3. Challenge: State Management and Contextual Accuracy

- **The Challenge**: For the LLM to be a useful "copilot," it needs the correct context for every query. This context includes three distinct layers: the general medical knowledge from the RAG system, the specific patient's records, and the history of the current conversation. Managing this state, especially when a doctor switches between different patients, was critical.
- **The Solution**: We leveraged Streamlit's session state (`st.session_state`) to manage the application's memory. When a doctor selects a patient from the dropdown menu, the application checks the session state. If the selected patient is new or different from the one in the current session, the system purges the old chat history and loads the

history for the newly selected patient from our `patients.json` file. The LangChain chain was constructed to explicitly accept these multiple context inputs (`rag_context`, `patient_context`, `chat_history`), ensuring the final prompt sent to the LLM is always comprehensive and accurate.

---

4. Challenge: Performance Optimization and Model Selection

- **The Challenge**: Our initial plan was to use a single, powerful, state-of-the-art multimodal model (e.g., a large model like Llama 3.2 Vision) to handle all tasks. However, early testing on local hardware revealed that such large models had significant resource requirements, leading to slow inference times. This lag made the application feel unresponsive and impractical for the real-time, interactive experience a "copilot" needs.
- **The Solution**: We pivoted from a "one-model-fits-all" approach to a more optimized, multi-model strategy. We selected smaller, more specialized models that were better suited for their specific tasks, significantly improving performance:
    - **For Text Analysis**: We chose **`llama3:8b`**. This model provides an excellent balance of strong language comprehension and fast performance, making it ideal for the text-based RAG queries that form the bulk of the interactions.
    - **For Image Analysis**: We selected **`llava:7b`**. This is a mature and highly efficient vision model specifically designed for tasks like image description and analysis. By using it only when an image is uploaded, we avoid the computational overhead of a large multimodal model during text-only conversations.

This strategic decision to use specialized, right-sized models was the most critical performance optimization, resulting in a fast, responsive, and highly usable application.