Project Report: A Domain-Specific Conversational Assistant for Healthcare

Table of Contents

---

1. Executive Summary

This project addresses the task of building a domain-specific conversational assistant for a healthcare organization. The primary goal was to create a system capable of safely answering general patient queries about symptoms and health guidelines, while strictly avoiding the provision of medical advice and deflecting emergency situations.

After a comparative analysis of several model engineering techniques—including full fine-tuning, Parameter-Efficient Fine-Tuning (PEFT) with LoRA, and advanced prompt engineering—the chosen approach was to leverage a powerful foundation model, **Google's Gemini 1.5 Flash, via its API**. This method was selected for its rapid development cycle, state-of-the-art capabilities, and the ability to build a robust solution without requiring specialized GPU hardware for training.

A **three-layer safety framework** was implemented as a core component of the system, consisting of an input guardrail for emergency detection, a prompting guardrail to define the AI's behavior, and an output guardrail to ensure safety disclaimers are always present. An interactive web application was developed using **Gradio**, allowing for real-time experimentation with Zero-Shot, Few-Shot, and Chain-of-Thought (CoT) prompting techniques. The final application was containerized using **Docker** for reproducibility and designed for easy deployment on platforms like Hugging Face Spaces.

Evaluation confirmed that the system effectively recognizes and blocks sensitive queries. The Chain-of-Thought prompting technique proved to be the most reliable method, producing reasoned, relevant, and safe responses that consistently adhere to the defined constraints.

2. Problem Definition and Objectives

2.1. Business Need

Healthcare organizations are often inundated with patient queries ranging from simple administrative questions to complex symptom descriptions. A conversational AI can serve as a first line of support, providing instant, 24/7 access to general health information. This can reduce the load on human staff, improve patient engagement, and provide a valuable, accessible resource for non-emergency health questions.

2.2. Core Objectives

- **Develop a Conversational AI:** Build an assistant that can understand and respond to natural language queries about health.
- **Ensure Domain Specificity:** The AI's responses should be relevant to general healthcare, symptoms, and wellness guidelines.

- **Prioritize Safety:** The system must operate within strict safety protocols, recognizing its limitations as a non-medical professional.
- **Deploy as a Usable Application:** The final product should be an accessible web application with a user-friendly interface.

2.3. Critical Safety Constraints

- **No Medical Advice:** The AI is strictly forbidden from providing diagnoses, prescribing treatments, or interpreting medical results.
- **Flag Sensitive Queries:** The system must identify and refuse to answer queries that are medical emergencies or involve sensitive topics like self-harm.
- **Mandatory Disclaimer:** Every informational response must include a clear disclaimer advising the user to consult a healthcare professional.

3. Methodology and Technical Approach

A multi-faceted approach was considered for this project, with safety being the paramount concern at every stage.

3.1. The Three-Layer Safety Framework

A non-negotiable safety architecture was designed to operate independently of the chosen AI model.

1. **Input Guardrail:** A pre-processing step that scans user queries for a predefined list of high-risk keywords (e.g., "chest pain", "suicide"). If a keyword is detected, the query is blocked from reaching the AI, and a standardized emergency response is returned.
2. **Prompting Guardrail:** The prompt sent to the LLM is engineered to contain a detailed "system persona" that explicitly defines the AI's role, its limitations, and its strict rules of engagement, including the requirement to be harmless and provide a disclaimer.
3. **Output Guardrail:** A post-processing step that verifies the AI-generated response. If the mandatory disclaimer is missing for any reason, it is automatically appended to the end of the message before being shown to the user.

3.2. Model Engineering: A Comparative Analysis

Three primary techniques for adapting an LLM were evaluated.

- **Description:** This involves updating all weights of a pre-trained model on a domain-specific dataset.
- **Pros:** Can achieve the highest level of specialization and performance, embedding deep domain knowledge directly into the model's parameters.

- **Cons:** Extremely computationally expensive and memory-intensive. Fully fine-tuning even a "small" model (~77M parameters) is often infeasible without high-end, industrial-grade GPUs.
- **Description:** A more modern approach where the original model's weights are frozen. Small, trainable "adapter" layers (LoRA) are inserted into the model, and only these adapters are trained.
- **Pros:** Drastically reduces memory and compute requirements (by up to 90%), making fine-tuning accessible on consumer-grade GPUs. It offers a powerful balance between performance and efficiency.
- **Cons:** Still requires a GPU for training and a well-prepared dataset.
- **Description:** This technique involves using a powerful, pre-trained foundation model (like Google's Gemini) without any training. All specialization is achieved by crafting detailed, instructive prompts that guide the model's behavior at inference time.
- **Pros:**
    - **No Training Required:** Eliminates the need for GPUs and complex training pipelines.
    - **Rapid Prototyping:** Allows for extremely fast development and iteration.
    - **State-of-the-Art Foundation:** Leverages the immense knowledge of a massive, general-purpose model.
-
- **Cons:**
    - **Dependency on External API:** Relies on a third-party service, which may have associated costs and latency.
    - **Less Control:** The model's core knowledge cannot be altered; behavior is guided, not rewritten.
-

3.3. Data Sourcing and Usage

While the chosen prompt engineering approach does not require a training dataset, high-quality medical datasets like **lavita/ChatDoctor-HealthCareMagic-100k** and **bigbio/med_qa** were used as a reference to craft realistic and effective examples for the Few-Shot and Chain-of-Thought prompts. These datasets, containing real doctor-patient interactions and medical exam questions, are ideal sources for understanding the structure of medical queries and formulating safe, informative responses.

4. Implementation Details: The Gemini API with Prompt Engineering

4.1. Rationale for Selection

The Prompt Engineering approach (Approach 3) was selected for this project's implementation due to its accessibility, speed, and ability to produce a highly effective and safe prototype without specialized hardware. It allowed for a focus on the most critical aspects: safety guardrails and user interface design.

4.2. Prompt Engineering Laboratory

An interactive application was built to experiment with three prompting techniques:

- **Zero-Shot Prompting:** A direct instruction defining the AI's role and task.
- **Few-Shot Prompting:** The instruction is augmented with 2-3 high-quality examples of questions and desired answers, helping the model learn the expected format and tone.
- **Chain-of-Thought (CoT) Prompting:** The most advanced technique. The prompt includes an example where the AI "thinks step-by-step" to deconstruct a query, formulate a safe answer, and add the disclaimer. This forces a more logical and cautious reasoning process.

4.3. Core Inference Logic

The application's core logic is a Python function that:

1. Receives the user's query and the selected prompting method.
2. Passes the query through the **Input Guardrail**.
3. Constructs the appropriate prompt (Zero-Shot, Few-Shot, or CoT).
4. Sends the prompt to the Gemini API using the genai.Client().
5. Receives the response and passes it through the **Output Guardrail** to ensure the disclaimer is present.
6. Returns the final, safe response to the user.

5. Model Evaluation

5.1. Intent Recognition (Safety Guardrail Evaluation)

The Input Guardrail was tested with a suite of sample queries. It demonstrated **100% accuracy** in identifying and blocking queries containing explicit emergency keywords.

Query	**Keyword Detected**	**Action Taken**
"What are the symptoms of a cold?"	None	Process
"I am having severe chest pain"	chest pain	Block
"I feel suicidal"	suicidal	Block

5.2. Response Relevancy and Safety (Qualitative Analysis)

A qualitative comparison of the prompting techniques revealed a clear hierarchy of performance:

- **Zero-Shot:** Often provided correct information but was inconsistent. It occasionally forgot to include the disclaimer or produced responses that were too brief.

- **Few-Shot:** Showed a significant improvement. Responses were well-structured, matched the tone of the examples, and reliably included the disclaimer.
- **Chain-of-Thought (CoT):** Produced the highest quality responses. The answers were more detailed, logical, and demonstrably cautious. By forcing a reasoning step, the model was less likely to make declarative statements and more likely to frame information in a general, safe context. **CoT was selected as the default and recommended method.**

6. Deployment Architecture

6.1. Technology Stack

- **AI Service:** Google Gemini API (gemini-1.5-flash)
- **Web Framework:** Gradio (for a fast, interactive UI)
- **Backend API (Optional):** FastAPI (integrated by Gradio)
- **Containerization:** Docker
- **Hosting Platform:** Hugging Face Spaces (Ideal for Gradio apps)

6.2. Deployment Workflow

A diagram of the final application flow:

```
[User] <--> [Gradio Web UI] <--> [Python Backend (main.py)]

                                   |

                                   V

                           [1. Input Guardrail Check]

                            (Is it an emergency?)

                                   |

                                   V (If Not)

                          [2. Construct CoT Prompt]

                                   |

                                   V

                             [3. Call Gemini API]

                                   |
          V

[4. Output Guardrail Check]

  (Add disclaimer if needed)

          |

          V

[Return Response to Gradio UI]
```

The entire application, including the Python environment and dependencies, is encapsulated in a **Docker image**. This image can be run locally or deployed to any cloud platform, ensuring perfect reproducibility. Hugging Face Spaces was chosen as the ideal deployment target due to its seamless integration with Gradio and its secure handling of API keys as repository secrets.

7. Conclusion and Future Work

This project successfully demonstrates that a highly effective and, most importantly, safe domain-specific AI assistant can be built without the need for model training, by leveraging a powerful foundation model API and robust prompt engineering. The implemented three-layer safety framework ensures that the assistant operates within its defined boundaries, reliably deflecting emergencies and providing necessary disclaimers.

Future Work:

- **Hybrid Approach:** While the prompt-engineered solution is excellent, the next step would be to apply **fine-tuning with LoRA** on a dataset like ChatDoctor-HealthCareMagic-100k. This would create a fully self-hosted model with deeper, more ingrained domain expertise, potentially leading to even more nuanced and consistent responses.
- **Advanced Guardrails:** The keyword-based input guardrail could be enhanced by training a separate, small text classification model to detect user intent (e.g., "seeking information" vs. "describing emergency") for more sophisticated query flagging.
- **Knowledge Base Integration (RAG):** To provide answers based on an organization's specific, private documents (e.g., internal treatment guidelines), a Retrieval-Augmented Generation (RAG) pipeline could be implemented.