

DOCTOR FINDER CHATBOT

For completing this project, the approach involved structured web scraping to collect clean hospital data, followed by extensive feature engineering to normalize and enrich it. A hybrid query-handling layer was then built using fine-tuned BERT and fuzzy matching, ensuring both semantic depth and robustness. Finally, the solution was deployed as an interactive Streamlit app with intuitive filters and a hospital-friendly UI, making doctor discovery fast and user-centric.

WEB SCRAPING

Web scraping was the part which challenges were there and did it, skipped the HTML-parsing route and leaned on **Requests/HTTPX** to hit the same XHR endpoints that the hospital portals themselves were calling behind the scenes. That gave clean JSON payloads instead of messy HTML.

- Inspect Network tab
- **Requests/HTTPX**
- Parse JSON
- Normalization

FEATURE ENGINEERING

After getting the data feature engineering was done successfully where the raw XHR fetched JSON data was inconsistent and messy. Engineered features to make the downstream model and filters robust, interpretable, and user-friendly.

- **Specialty Normalization:** Created a canonical mapping dictionary and Applied **dual normalization**, on the **data side** (scraped specialties) and on the **query side** (user input synonyms, abbreviations)
- **Fee Range Engineering :** Converted fee strings into numeric min/max, buckets, and median values.
- **Timing Features:** Parsed inconsistent time formats ("10am-1pm", "10:00–13:00", "Morning") into **start-time** and **end-time** in 24-hour format.
- **Textual Features:** Cleaned doctor names (removed honorifics like "Dr.", "Prof." → stored separately as title) tokenized and lowercased specialties for embedding-based similarity
- **Location Features:** Standardized addresses into city/locality/pincode.
- **Bias-Correction Features:** Introduced **frequency counts** of specialties to detect underrepresented classes, with balancing applied during training.
- **Caching:** Pre-computed embeddings for specialties and clinic names to speed up retrieval.

MODEL INTEGRATION AND QUERY HANDLING

After the feature engineering stage, moved into **model integration and fine-tuning** so the chatbot could actually leverage those engineered signals

- **Base Model:** Used a fine-tuned BERT variant for **intent classification** and **specialty matching**. Trained on augmented data with synonyms and paraphrases to improve generalization. Objective is multi-class classification for intent and multi-label classification for specialties. Provides semantic depth, allowing the chatbot to interpret more complex or conversational queries.
- **Fuzzy Matching & Rule-Based Parsing:** Matches user input against normalized specialty names, handling spelling errors and abbreviations. Synonym/replacement dictionary maps colloquial terms (e.g., “heart doctor” → “Cardiology”). Regex-based parsing extracts fee ranges (“under 600”, “between 400–700”) and availability keywords (“evening”, “weekend”). Acts as a fallback when model confidence is low, ensuring the chatbot always returns meaningful results.
- **Hybrid Pipeline Assembly:** Input → Pre-processing → BERT Inference → Confidence Check
 - If confidence is high → use BERT predictions.
 - If confidence is low → fallback to fuzzy matching + engineered feature filtering.
 - Final results are ranked and displayed through the Streamlit interface.

This hybrid design balances **accuracy** (via BERT) with **resilience and speed** (via fuzzy matching), making the chatbot both intelligent and reliable in real-world use.

STREAMLIT INTERFACE AND FILTERS

The final stage was creating and deploying the chatbot interface using Streamlit. The application was designed with a clean, hospital-friendly layout featuring gradient headers, emoji-framed section titles, and clear micro-copy to guide users. Interactive filters allowed patients to search by specialty and fee range.

Sidebar Filters

- **Find by Specialty:** Dropdown menu populated with normalized specialty names. Patients can directly select the specialty they are interested in (e.g., *Cardiology, Orthopedics, ENT, etc.*).
- **Sort by Fee:** Radio buttons to sort results either **Low** → **High** or **High** → **Low**, giving flexibility in how doctors are ranked.
- **Results Display:** Doctors matching the filters are displayed in a clean tabular format with details such as **Name, Specialty, Fee, Timings, and Clinic Location, Contact Number**. Sorting logic dynamically reorders the results based on the selected fee preference. Specialty and fee filters work in combination with the hybrid query-handling pipeline (BERT + fuzzy matching), ensuring accurate and user-friendly results.

This design ensures that patients can **search by specialty, filter by budget, and sort results according to preference**, all within a simple and responsive Streamlit dashboard.

NOTE:

In this project, the chatbot uses a mix of two methods. First, every user query is passed through the fine-tuned DistilBERT model, which predicts what the user is asking for (like finding a doctor by specialty, checking fees, or availability). The label encoder then converts this prediction into a readable intent name. Based on this intent, the chatbot decides which part of the code should run. Alongside this, fuzzy matching and simple rules are used to catch spelling mistakes, short forms, and common phrases. If the model is confident, its result is used; if not, the system falls back to fuzzy matching and filters. This way, the chatbot combines the accuracy of the model with the reliability of rule-based checks, making it both smart and practical.