SMART TRAVEL ASSISTANT

By Aayush Sharma

Approach:

1. Intent-aware routing

   Every user message is first normalised (case-folded and de-noised) and passed through a tiny linguistic classifier that checks for three independent clues: the presence of fare-related keywords, a plausible "origin–destination–date" phrase, and an explicit calendar date. Only when all three signals concur is the query flagged as a price-prediction request; everything else is treated as a policy question. The check is entirely rule-based, runs in microseconds on a single CPU core, and guarantees that numerical inference and document retrieval remain strictly separated—an essential property for predictable latency and auditability.

2. Retrieval-Augmented Generation for policy related questions

   Queries routed to the policy branch are answered by a classical RAG pipeline. Airline manuals and fee schedules are pre-processed into paragraph-sized chunks, converted into dense semantic vectors with a sentence-transformer, and stored in a flat FAISS index so that similarity search reduces to one dot-product operation per chunk. At runtime the question is embedded, the most relevant chunks are fetched, and their scores are slightly boosted when they share exact phrases with the question to improve lexical faithfulness. The top three chunks are then injected into an instruction prompt for a compact, CPU-friendly Llama language model. The model is asked to draft a sub-50-word answer that relies only on the supplied context; if no chunk contains the answer it must respond with *information not found*. The reply ends with a citation that names the source document and chunk identifier, providing immediate traceability.

3. Gradient-boosted fare prediction with post-hoc explanation

   For price requests the system converts free-text into structured features: airline, origin, destination, travel date, flight duration, number of stops, historical demand for the month, and a fuel-consumption proxy derived from duration and stops. These variables feed a

Bayesian-optimised gradient-boosted decision-tree regressor that produces a fare estimate in a few milliseconds. Immediately afterwards, a model-specific SHAP explainer decomposes the prediction into the model's baseline expectation plus per-feature contributions. The traveller receives the quoted fare, the global baseline, the observed historical price band for that route (or a conservative ±10 % band when no history exists), and a concise natural-language summary of the top factors that pushed the fare up or down, accompanied by a waterfall plot for visual inspection.

4. Unified conversational interface
   Both the RAG and prediction branches return their results to a Streamlit-based chat front-end that stores the entire dialogue in session memory. Policy answers appear as short, cited paragraphs, while fare estimates appear as formatted fare cards followed by the factor breakdown and embedded explanatory graphic. Because routing is deterministic and the two engines share no state, either branch can be upgraded—larger language model, newer fare dataset, GPU acceleration—without rewriting the other, yielding a low-latency, transparent, and easily extensible travel assistant.

Metrics:

The gradient-boosted regressor explains roughly 92 % of the variation in ticket prices ($R^2 \approx 0.92$) while holding the mean absolute error to about ₹ 675 per ticket—comfortably below the day-to-day swings seen on busy domestic routes. This performance comes from an ensemble of 1 430 trees of depth 7, trained with a modest learning rate (~0.05), high-fidelity sampling (subsample ≈ 0.96, colsample_bytree ≈ 0.65), and strong regularisation ($\lambda \approx 5.8$, $\gamma \approx 3.6$) to suppress over-fitting. The resulting model achieves a practical balance between accuracy and generalisability, delivering fare estimates that are typically within a few hundred rupees of reality.

Challenges:

Building the assistant meant taming three stubborn problems. First, travellers phrase requests chaotically-"BLR → DEL on 15 Aug", "next Friday's price to Bombay", or "how much will it cost me?"-so the router had to stay ultra-fast yet still separate fare queries from policy questions without constant misfires. Second, the source PDFs are an inconsistent mess: crisp tables from one airline, blurry OCR scans from another; extracting clean, chunk-sized text that every embedder could understand required just enough cleansing to boost recall without adding latency. Third, the fare model had to cope with lopsided data-plenty of records for trunk routes such as BOM-DEL, but almost none for secondary city-pairs. We reduced the risk of wild extrapolation by injecting demand-aware signals (monthly flight volume, route length, fuel-proxy) and by training with strong regularisation so the model defaults to airline-agnostic baselines when evidence thins out. At inference time we also detect unseen origin-destination pairs and wrap the quote in a ±10 % safety band marked "low-confidence," preserving usefulness without pretending to a precision the data cannot support.

Improvements:

Looking ahead, the most immediate upgrade is to replace the current rule-based router with a lightweight intent classifier trained on actual chat logs. A small Distil-BERT fine-tuned on a few hundred labelled queries would wipe out the edge-case hand-off errors we still see—"cheap fare to Goa?" being routed to RAG, for example—yet adds only a handful of milliseconds of latency thanks to on-device quantisation.

Interpretability can also be improved. Today's SHAP read-out exposes raw one-hot columns such as cat__Airline_IndiGo, which are accurate but cryptic. Mapping those technical features into human-friendly composites ("red-eye departure", "fuel-heavy multi-stop", "holiday-month surge") and grouping them before attribution would let travellers understand why a fare is high at a single glance rather than parsing a waterfall plot full of engineering labels.

Finally, to broaden appeal we plan to layer a full itinerary engine on top of the existing pipelines. The idea is to let a user specify a destination and travel window, then generate a day-by-day plan with flight quotes from our model, hotel rates pulled from a partner API, and a running budget tally. The itinerary text itself can be produced by the same LLM family we already host, while cost aggregation reuses the fare predictor and a slim hotel-pricing micro-service. Together these upgrades would make the assistant more accurate, more transparent, and far more useful for end-to-end trip planning.