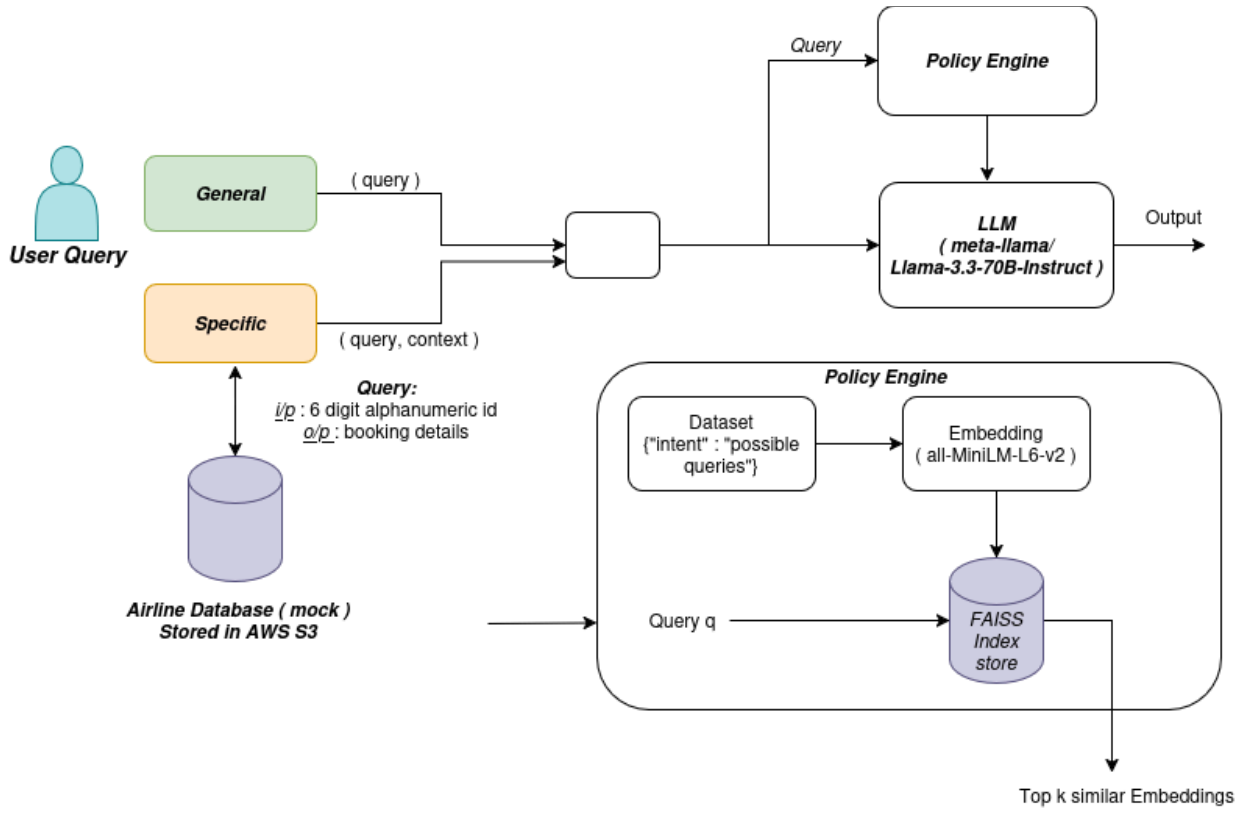

ABSTRACT

This project presents the design and implementation of an intelligent customer support chatbot for airline operations. The system leverages a fine-tuned large language model (Llama-3.3-70B-Instruct) enhanced with a retrieval-augmented generation (RAG) mechanism utilizing similarity search via a FAISS vector index. Contextual knowledge and historical data are stored securely in AWS S3, while model deployment and inference are managed through Nebius AI Studio. The chatbot efficiently interprets passenger queries, retrieves the most relevant contextual information, and generates coherent, accurate free-text responses.

OBJECTIVE

The primary goal is to automate airline customer interaction, ensuring real-time query handling, reduced support workload, and consistent service quality. The chatbot is optimized to:

1. Understand complex natural language queries.
2. Retrieve semantically relevant airline policy or flight information.
3. Generate human-like responses based on retrieved data and fine-tuned model knowledge.

4. Operate efficiently within a managed AI infrastructure (Nebius AI Studio).

SYSTEM ARCHITECTURE OVERVIEW

The overall system architecture consists of five core modules:

1. **User Interaction Layer:** Handles user requests through front-end interfaces or APIs. Inputs are passed as raw text queries.

2. **Policy Engine:** Determines the intent and retrieves relevant data embeddings from the FAISS vector store. It employs similarity search using cosine distance to find the most semantically related context vectors.

3. **FAISS Vector Index Store:** A static repository (~300 entries) containing vector embeddings generated using the `all-MiniLM-L6-v2` model. Each vector corresponds to airline FAQs, policy documents, and operational texts.

4. **AWS S3 Storage:** Hosts the original structured and unstructured knowledge base (documents, tickets, logs). S3 ensures persistence, scalability, and availability of reference data used during index generation.

5. **LLM Core (Llama-3.3-70B-Instruct):** The core reasoning unit, fine-tuned using the Parameter Efficient Fine-Tuning (PEFT) method to specialize on airline-specific conversational tone and terminology.

These modules interact through internal function calls rather than external APIs, ensuring low-latency and controlled communication between components.

DATA FLOW DESCRIPTION

1. **Input Stage:**
   The user submits a query (e.g., “Can I change my flight date after check-in?”).

2. **Policy Engine Processing:**

   - The query is converted into an embedding vector using the same embedding model as the FAISS index (`all-MiniLM-L6-v2`).

   - The FAISS store performs similarity search to find the top-k (typically k=5) most similar context vectors.

    - The retrieved textual contexts are aggregated into a context package.

3. **Context Injection into LLM:**
    - The context package and user query are concatenated into a structured prompt template.
    - This prompt is sent to the Llama-3.3-70B-Instruct model hosted in Nebius AI Studio.
4. **Response Generation:**
    - The fine-tuned model generates a free-text response.
    - No rigid schema or JSON structure is enforced to allow flexible conversational tone.
5. **Output Delivery:**
    - The response is displayed to the user.
    - Optional logging can be done to S3 for audit or retraining purposes.

MODEL TRAINING DETAILS (PEFT)

The Llama-3.3-70B-Instruct model underwent **Parameter Efficient Fine-Tuning (PEFT)**.

- Only a small subset of adapter weights was trained, reducing GPU memory requirements and preserving general reasoning capabilities.
- Domain data used for fine-tuning included airline FAQs, policy texts, and real customer service transcripts.
- PEFT allowed specialization without catastrophic forgetting of base model general language skills.

The objective function minimized loss over conversational accuracy and context relevance, while inference used Nebius AI Studio's managed GPU infrastructure for scalability and stability.

EMBEDDING AND SIMILARITY SEARCH LOGIC

The similarity search relies on vectorized document representations. Each document is transformed into a high-dimensional vector (dimension ≈ 384 for `all-MiniLM-L6-v2`).

Mathematically:
Let `q` be the query embedding and `D = {d1, d2, …, dn}` be stored document vectors.
The similarity score for each document `di` is computed as:

```
score(q, di) = cos(q, di) = (q · di) / (||q|| * ||di||)
```

The FAISS index maintains these vectors in an optimized search space using inner product quantization to enable millisecond-level lookup for k-nearest neighbors.

DEPLOYMENT OVERVIEW

- **Model Hosting:** Managed via **Nebius AI Studio**, which abstracts GPU provisioning, scaling, and endpoint management.
- **Data Storage:** AWS S3 buckets store both the raw corpus and generated embeddings for version control and reproducibility.
- **Index Maintenance:** FAISS index is static; updates are infrequent and performed manually when new policy data is added.
- **Access Control:** Internal IAM policies ensure only authorized services access S3 and model endpoints.

SYSTEM ADVANTAGES

- **Scalability:** Managed inference via Nebius allows elastic scaling with user demand.
- **Performance:** Similarity search reduces LLM token load by injecting only relevant context.
- **Domain Adaptation:** PEFT fine-tuning ensures domain-specific accuracy without retraining full model parameters.
- **Data Security:** S3 and Nebius access control enforce isolation and confidentiality.

LIMITATIONS

- Static FAISS index requires manual updates.
- Absence of real-time evaluation metrics may hinder continuous optimization.
- Latency depends on Nebius inference queue times.
- No explicit reinforcement learning loop for performance improvement.

FUTURE ENHANCEMENTS

- Integrate active learning to automatically expand FAISS vectors with new query-answer pairs.
- Introduce lightweight evaluation metrics for accuracy and latency tracking.
- Deploy a caching layer for repeated queries.
- Explore hybrid retrieval combining FAISS with keyword-based search for better coverage.

CONCLUSION

The developed architecture demonstrates a production-ready, modular, and efficient design for an airline customer support chatbot. It combines semantic retrieval, efficient large-model reasoning, and secure cloud storage, yielding a high-quality conversational agent adaptable for enterprise-scale deployment.