Jupiter Help Centre FAQ Chatbot

Introduction

The Jupiter Help Centre FAQ Chatbot is designed to deliver a better, more conversational self-service experience for users of Jupiter Money. Jupiter's official Help Centre contains static FAQ pages covering topics like payments, cards, KYC, rewards, and transaction limits. However, these pages require users to manually navigate categories or search using exact keywords, which is not always intuitive or user-friendly.

This project builds an intelligent chatbot that allows users to ask natural, free-form questions—potentially in English, Hindi, or Hinglish—and receive clear, friendly answers pulled from Jupiter's existing FAQs. The chatbot combines automated scraping, data cleaning, semantic similarity search, and optional LLM-based rephrasing to deliver high-quality answers.

Objective

The main objective is to create an intelligent FAQ assistant that can:

- Automatically extract and maintain Jupiter's official FAQs.
- Understand user queries even when phrased differently from the original FAQ.
- Retrieve the most relevant existing answer confidently.
- Optionally rephrase the answer in a clear, friendly, conversational tone.
- Support multilingual queries in English, Hindi, and Hinglish.

Problem Statement

Users often want quick, direct answers without reading long FAQ pages or navigating static sections. They may ask questions in different ways that don't match the exact phrasing of the FAQ entries, leading to search failures or user frustration.

A static FAQ page also does not handle multilingual variations or conversational expectations. This gap leads to support tickets or user churn, which could be improved with an intelligent, user-friendly FAQ bot.

High-Level Solution

The project uses a streamlined pipeline with these stages:

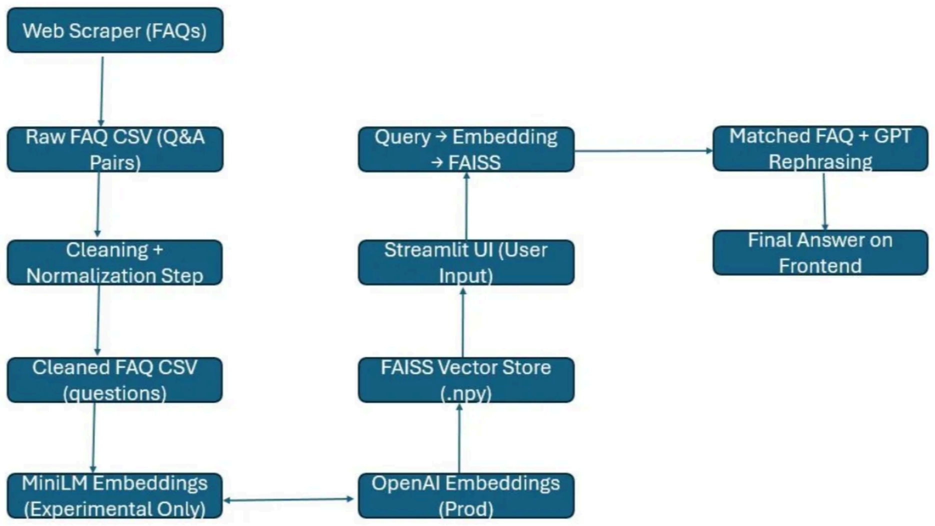
- **Scraping**: Automatically collects the latest FAQs from Jupiter’s Contact page.
- **Cleaning**: Normalizes text, removes noise, deduplicates entries.
- **Embedding Generation**: Converts questions into semantic vectors using local or API-based models.
- **Semantic Search**: Finds the closest FAQ match using cosine similarity or FAISS indexing with threshold filtering.
- **LLM-Based Rephrasing (Optional)**: Uses GPT to generate clear, conversational answers.
- **Evaluation**: Compares retrieval-only and LLM-enhanced answers for accuracy and latency.
- **Streamlit App**: Provides a multilingual, user-friendly interface with related question suggestions.

System Architecture Overview

The system follows a modular pipeline:

- **User Query Input**: The user asks a question in natural language.
- **Embedding Generation**: The query is transformed into a semantic vector using a pretrained model or OpenAI’s embedding API.
- **Similarity Search**: The system searches a store of precomputed FAQ question embeddings to find the closest match using cosine similarity or FAISS.

- **Threshold Filtering**: If the best match exceeds a similarity threshold, it is considered valid; otherwise, the system gracefully declines.
- **LLM Rephrasing (Optional)**: The retrieved answer can be passed to GPT to produce a friendlier, clearer response.
- **User-Facing Output**: The final answer is shown to the user, along with optional suggestions for related questions.


Web Scraper (FAQs)
Raw FAQ CSV (Q&A Pairs)
Cleaning + Normalization Step
Cleaned FAQ CSV (questions)
MiniLM Embeddings (Experimental Only)
OpenAI Embeddings (Prod)
FAISS Vector Store (.npy)
Streamlit UI (User Input)
Query → Embedding → FAISS
Matched FAQ + GPT Rephrasing
Final Answer on Frontend

Methodology

Data Collection

The first step was to scrape Jupiter's public Contact page, which contains structured FAQ question–answer pairs. The scraper was designed to automatically target the correct sections of the HTML page, extracting text while ignoring formatting noise and irrelevant elements.

Cleaning and Preprocessing

After scraping, the raw FAQ entries often contain inconsistent spacing, stray symbols, or duplicated questions. A cleaning pipeline was applied to normalize whitespace, remove special characters and tags, deduplicate entries, and filter out entries that were too short or incomplete to be useful.

The result is a clean, well-structured dataset of FAQs stored as a CSV file, ready for semantic modeling.

Semantic Embedding Generation

Each question in the cleaned FAQ dataset is converted into a high-dimensional vector (embedding) that captures its semantic meaning. Two embedding approaches are supported:

- **Local**: Using SentenceTransformer models like `all-MiniLM-L6-v2`, suitable for offline or open-source deployments.
- **API-based**: Using OpenAI's Embedding API for more consistent multilingual and high-quality embeddings.

These embeddings are saved in a binary format for efficient reuse.

Similarity Search and Indexing

When a user submits a query, it is converted into an embedding using the same model or API. Semantic similarity is then computed between the query and all stored FAQ embeddings.

Two retrieval modes are supported:

- **Cosine Similarity (NumPy)**: Simple, effective for small datasets.
- **FAISS Index**: Highly efficient for large datasets, allowing scalable, sub-second retrieval times even with thousands of FAQs.

A threshold is applied to ensure that only high-confidence matches are returned.

LLM-Based Rephrasing

To further improve user experience, the system optionally passes the retrieved FAQ answer to an OpenAI GPT model (e.g., GPT-3.5-turbo). The prompt is carefully designed to preserve the factual content while rephrasing it into clear, friendly, conversational language. This step supports multilingual input and output, making it accessible for users in English, Hindi, or Hinglish.

Evaluation and Comparison

The system was tested on diverse queries in English, Hindi, and Hinglish. Both retrieval-only and LLM-based approaches delivered high accuracy, consistently returning correct and complete answers.

User experience differed between methods. Retrieval-only responses were fast and precise but sometimes felt too brief or formal. LLM-based answers were more conversational and user-friendly, better suited for users less familiar with digital banking.

Latency was also a consideration. Retrieval was nearly instantaneous, ideal for high-traffic scenarios or systems avoiding external API calls. LLM responses, while slower due to OpenAI API use, remained acceptable for most support contexts given their improved clarity and tone.

Both methods handled multilingual input effectively. While the LLM occasionally paraphrased or shortened responses, careful prompt design ensured essential details were preserved.

Streamlit Application

An additional Streamlit application is developed to provide an interactive, user-friendly front-end for the chatbot. The app supports:

- Inputting questions in English, Hindi, or Hinglish.
- Entering an OpenAI API key to enable embedding generation and LLM rephrasing.
- Automatic embedding caching to avoid repeated API calls.
- Building and using a FAISS index for fast retrieval.
- Displaying both retrieval-only and LLM-rephrased answers side-by-side with similarity scores and latency.
- Suggesting related FAQ questions to help users discover more answers.

This web interface makes the system accessible for demos, internal testing, and potential customer-facing deployment.

Features Summary

- Automated scraping of Jupiter’s FAQ content.
- Robust cleaning and normalization pipeline.
- Semantic embedding generation using local or API-based models.
- Fast, scalable retrieval with FAISS indexing.
- LLM-based rephrasing for friendly, clear responses.
- Multilingual query support in English, Hindi, and Hinglish.
- Streamlit web interface for real-time interaction.
- Logging and evaluation of similarity scores and latency.

Conclusion

This project demonstrates a complete pipeline for transforming static FAQ content into an intelligent, interactive chatbot. By combining automated data extraction, semantic search, and powerful language models, it allows users to ask questions naturally and receive accurate, user-friendly answers—reducing friction and improving self-service support experiences.

The system is modular, extensible, and designed to handle real-world variations in phrasing and language. With further refinement, it can be deployed as a production-ready FAQ assistant for fintech apps or any service with large, evolving knowledge bases.