**Building a Human-Friendly FAQ Bot from Jupiter Help Centre**

**1. Introduction**

This project aimed to address a common user need: quick, conversational answers to frequently asked questions, bypassing the need to navigate static help pages. Specifically, the objective was to develop an intelligent chatbot for Jupiter's Help Centre by scraping existing FAQs and enabling a more intuitive question-answering experience. The solution leverages modern natural language processing (NLP) techniques, including semantic search and text rephrasing, wrapped in an interactive user interface.

**2. Problem Breakdown & Solution**

**2.1. Scrape the FAQs**

**Objective:** Crawl the FAQ section of the Jupiter help website (https://jupiter.money/contact/) and structure the data into categorized question-answer pairs.

**Methodology**:

1) The requests library was used to perform an HTTP GET request to the specified URL, fetching the HTML content of the page.

2)BeautifulSoup4 was then employed to parse this HTML. By inspecting the webpage, it was identified that the FAQ items were contained within a <ul> tag with the attribute data-controller='faq'.

3) Each FAQ item (a <li> tag within this <ul>) contained the question in a <span> tag and the answer in a <p> tag.

4) The text content from these <span> and <p> tags was extracted using .get\_text(strip=True) to remove leading/trailing whitespace and HTML tags.

5) These extracted question-answer pairs were stored in a list of dictionaries and subsequently converted into a Pandas DataFrame for easy manipulation and analysis.

### **2.2. Preprocess and Clean**

* **Objective**: Normalize and deduplicate similar questions, clean any HTML/formatting noise, and categorize content.
* **Methodology**:
  + **Text Cleaning**: A clean\_text function was implemented using regular expressions (re.sub(r'\s+', ' ', text)) to replace multiple whitespace characters with a single space and strip leading/trailing spaces, ensuring consistent formatting.
  + **Deduplication**: A new column, question\_cleaned, was created by applying lowercase conversion, removal of non-alphanumeric characters, and the clean\_text function to the original questions. The DataFrame was then deduplicated based on this question\_cleaned column, ensuring that semantically identical (after cleaning) questions were not repeated.
  + **Categorization**: A categorize function was developed to assign broad topics (e.g., 'KYC', 'Rewards', 'Payments', 'Limits', 'General') to each question based on the presence of predefined keywords in the question text (case-insensitive). This aids in organizing the knowledge base and could be used for filtering or improving search in more complex systems.

### **2.3. Build the FAQ Bot**

* **Objective**: Utilize language models to rephrase answers, handle semantically similar queries, and manage confidence in responses.
* **Methodology**: **Leveraging Language Models (LLMs - Specialized Transformer Models)**:

While not utilizing extremely large, generative LLMs like GPT-3 or LLaMA 3 for free-form text generation in every response, the project effectively employed **specialized transformer models** to achieve the desired intelligence:

1. **prithivida/parrot\_paraphraser\_on\_T5 (for Answer Rephrasing)**: This model, based on the T5 architecture, was loaded using AutoTokenizer and AutoModelForSeq2SeqLM. Its purpose was to rephrase answers into more conversational and natural language. Although the final bot implementation directly returns the original answer for simplicity and to prevent potential hallucinations from paraphrasing in a production environment, the capability is integrated into the code (paraphrase\_answer function) and can be activated if a dynamically .
2. **all-MiniLM-L6-v2 (for Semantic Similarity)**: This SentenceTransformer model is crucial. It converts both the scraped FAQ questions and user queries into high-dimensional numerical vectors (embeddings). These embeddings capture the semantic meaning of the text. This is fundamental for handling "semantically similar queries," allowing the bot to understand the intent of a user's question even if the exact phrasing isn't in the FAQ database.

**Embedding-Based Semantic Search with FA**

1. After generating embeddings for all cleaned FAQ questions, a FAISS (Facebook AI Similarity Search) index (faiss.IndexFlatL2) was created. FAISS is a library for efficient similarity search and clustering of dense vectors.
2. The IndexFlatL2 type was chosen for its simplicity and effectiveness, as it performs a brute-force search based on L2 (Euclidean) distance between vectors. Given that sentence embeddings are often normalized, L2 distance is proportional to cosine similarity, which is a standard metric for semantic similarity.
3. When a user query comes in, its embedding is generated using all-MiniLM-L6-v2, and then index.search() is used to quickly find the closest matching FAQ question in the database.

**Confident Responses & Graceful Declines**:

1. The find\_best\_faq\_match function incorporates a threshold parameter (defaulting to 0.4 for L2 distance). This threshold is critical for managing the bot's confidence.
2. If the distance between the user's query embedding and the best-matching FAQ embedding is *below* this threshold, it signifies a strong semantic match, and the corresponding answer is returned.
3. If the distance is *above* the threshold, it indicates a weak or no relevant match, and the bot gracefully declines by responding, "I'm not sure about that. Could you rephrase or ask something else?". This prevents the bot from providing irrelevant or incorrect answers.

### 

### **2.4. Demonstrate the Solution**

**Objective**: Develop a simple user interface to simulate user interactions.

**Methodology**:

* The Streamlit framework was chosen for rapidly building an interactive and shareable web application. Streamlit allows for creating data apps purely in Python, making it ideal for demonstration purposes.

**User Interface**:

st.title() and st.markdown() were used for descriptive headers.

st.chat\_input() provides an intuitive text input box for users to type their questions, mimicking a chat interface.

st.chat\_message("user") and st.chat\_message("bot") are used to display conversational turns, clearly distinguishing between user queries and bot responses.

**State Management**: st.session\_state was utilized to maintain the conversation history (st.session\_state.messages). This ensures that previous messages persist and are displayed even when the Streamlit app reruns due to user interaction.

**Performance Optimization**: The @st.cache\_resource decorator was applied to load\_and\_process\_faq\_data and load\_models\_and\_create\_index functions. This crucial optimization ensures that the time-consuming tasks of web scraping, model loading, and FAISS index creation are performed only once when the application starts, significantly improving responsiveness for subsequent user interactions.

**Interaction Flow**: When a user enters a question, it's added to the chat history. The find\_best\_faq\_match function is called to get the bot's response, which is then added to the history and displayed. The bot also gracefully handles 'exit' or 'quit' commands.

**3. Methodology and Architecture**

The architecture of the Jupiter FAQ Bot follows a typical Retrieval-Augmented Generation (RAG) pattern, though simplified. Instead of a generative LLM creating new answers, it retrieves the most relevant existing answer based on semantic similarity.

**Architectural Flow:**

**Data Acquisition (Scraping):**

* requests sends HTTP requests to the Jupiter contact page.
* BeautifulSoup parses the HTML to extract raw FAQ questions and answers.

**Data Preprocessing and Cleaning:**

* Pandas DataFrame is used for structured storage.
* Custom functions clean\_text perform normalization.
* Deduplication is performed on cleaned questions.
* Categorization function enriches the data with topic labels.

**Semantic Embedding:**

* The SentenceTransformer('all-MiniLM-L6-v2') model is used to convert all preprocessed FAQ questions into high-dimensional, semantically rich vector embeddings.

**Vector Indexing:**

* FAISS (IndexFlatL2) is used to create an efficient index of these embeddings. This allows for rapid similarity search.

**Query Processing and Retrieval:**

* When a user submits a query, it is also converted into an embedding using the same SentenceTransformer model.
* This query embedding is then used to search the FAISS index to find the FAQ question with the closest semantic meaning (lowest L2 distance).
* A confidence threshold is applied to this distance.

**Response Generation:**

* If a close match is found (distance below threshold), the corresponding original answer from the DataFrame is retrieved and returned.
* If no sufficiently close match is found, a predefined "I'm not sure" message is returned.
* (Optional, but present in code): prithivida/parrot\_paraphraser\_on\_T5 can be used to rephrase the retrieved answer for added conversational flow.

**User Interface (Streamlit):**

* A Streamlit application provides the interactive chat interface, displaying conversation history and accepting new user inputs.

**Why this Methodology?**

* **Accuracy and Control**: By retrieving answers directly from an existing FAQ knowledge base, the bot avoids the "hallucination" problem often associated with purely generative LLMs, ensuring answers are factual and align with Jupiter's official information.
* **Semantic Understanding**: Using SentenceTransformer and FAISS allows the bot to understand the user's intent even if the query is phrased differently from the original FAQ, providing a "human-friendly" experience.
* **Efficiency**: all-MiniLM-L6-v2 is a compact yet effective model, and FAISS provides lightning-fast similarity search, making the bot responsive without requiring massive computational resources.
* **Scalability**: The system can be easily scaled by adding more FAQs to the data source and rebuilding the FAISS index.
* **Cost-Effectiveness**: This approach avoids reliance on expensive API calls to very large commercial LLMs for every interaction, making it a more economical solution for an FAQ bot.
* **Demonstrability**: Streamlit provides a quick and effective way to showcase the bot's functionality interactively.

## **4. Deliverables**

* **A clean, well-documented notebook or application**:
  + The provided streamlit-faq-bot.py script serves as a complete, runnable, and well-commented application demonstrating the bot's functionality.
* **Evaluation of semantic similarity and relevance of answers**:
  + The core of the bot's intelligence relies on all-MiniLM-L6-v2 embeddings and FAISS for semantic similarity. The threshold parameter in find\_best\_faq\_match directly controls the relevance. A lower threshold requires a higher degree of similarity, leading to more confident (but potentially fewer) matches. Conversely, a higher threshold allows for more matches but might introduce less relevant answers.
  + For this assignment, the evaluation is demonstrated through interactive testing. For a production system, a more rigorous evaluation would involve:
    - **Test Dataset**: A curated set of user queries (some matching existing FAQs, some not, some paraphrased).
    - **Metrics**: Precision, Recall, F1-score (for retrieval effectiveness), and manual relevance judgments.
    - **User Feedback**: Collecting feedback on the helpfulness and accuracy of responses.
* **Clear documentation of methodology and architecture**:
  + This document itself fulfills this requirement by providing a comprehensive explanation of the project's design, implementation choices, and underlying principles.

## **5. Conclusion**

This project successfully developed a human-friendly FAQ bot capable of providing conversational and accurate answers to user queries related to Jupiter's services. By intelligently combining web scraping for data acquisition, robust preprocessing, specialized transformer models for semantic understanding and optional rephrasing, and an efficient FAISS-based retrieval system, the bot offers a superior user experience compared to traditional static FAQ pages. The Streamlit interface further enhances its usability, making it a practical and effective solution for improving user support.