ChatGPT

AI-Powered Customer Support Chatbot – Development Plan

1. Domain Selection

To choose the best domain, we evaluated available open datasets and real-world impact. **Travel/ Hospitality** emerged as the top choice. The Bitext "Travel LLM chatbot" dataset alone contains ~31,658 question–answer pairs covering common travel intents [1]. In contrast, open e-commerce FAQ data is very limited (e.g. a Kaggle set has only 79 Q/A pairs [2]), and public/healthcare data is sparse or sensitive. Travel chatbots are already widely used (for bookings, itineraries, etc.) and have proven utility in tourism [3]. Thus, focusing on a **travel customer-support assistant** (e.g. for airlines, hotels, or general travel queries) maximizes dataset support and practical value.

2. System Architecture (Retrieval+Generative)

We will use a **hybrid Retrieval-Augmented Generation (RAG)** architecture. Incoming user queries first trigger a retrieval step: relevant documents or FAQs are fetched from a knowledge base via vector search. Then, a generative LLM produces a response conditioned on the query plus retrieved context. This combines factual grounding (from the knowledge base) with fluent natural language responses. RAG-style systems improve accuracy by "referenc[ing] an authoritative knowledge base" alongside the LLM [4].

Key components: - **Knowledge Base & Index:** A collection of travel-related documents/FAQs is embedded using Sentence-Transformer models, and indexed with FAISS for fast similarity search [5]. - **Generative LLM:** An open-license large language model (e.g. Meta's LLaMA or TII's Falcon) is used to "understand and generate human-like responses." Such models can be fine-tuned on domain Q&A. For example, Meta's Llama-2 (commercial-friendly license) now leads open models [6], and fully open models like Falcon-7B, XGen, MPT-30B, etc. are available [7]. - **Chatbot Pipeline:** On each query, we embed the user's text, retrieve the top-k most relevant docs from FAISS, and feed them (with the query and any conversation history) into the LLM prompt. The LLM then generates an answer that is informed by the retrieved knowledge. This ensures up-to-date, factual replies while leveraging the LLM's language ability [4].

This hybrid design avoids "hallucinated" answers by grounding on real data, while still producing natural dialogue.

3. Datasets and Preprocessing

Primary Dataset: We will use the open *Bitext Travel LLM* dataset (31.6K Q/A pairs [1]). It covers travel intents like booking, baggage, cancellations, etc. This can be split into "question" vs. "answer" fields.

Supplementary Data: We may augment with general travel FAQs and guides. For example, scraping Wikipedia travel articles, tourist FAQ pages (e.g. airline/hotel support pages), or open tourism corpora.

Since all content must be open-source or public domain, we can use Wikipedia or Creative Commons travel guides.

Preprocessing Steps: - Clean text (remove HTML, normalize punctuation). - Replace templates/placeholders (like `{{Order Number}}`) with generic tokens or remove them. - For the **knowledge base**, concatenate related Q+A or documents into passages (~100–500 words each) to embed. Longer manuals or articles should be chunked. - For the **generative model**, format Q&A pairs into fine-tuning examples (e.g. prompt: "Q: ...? A: ..."). We may combine multiple Q's in one training example for variety. - (Optional) Label or categorize intents if using a simple intent classifier fallback. - Ensure data is deduplicated and balanced across categories.

All data and processing code will be stored in a shared repository. We will use Hugging Face Datasets or Pandas to load/manipulate the data.

4. Tools, Libraries, and Models

- **Embeddings & Vector Index:** SentenceTransformers (for text embeddings) and **FAISS** (open-source similarity search library) [5]. FAISS (by Facebook) efficiently indexes dense vectors for fast nearest-neighbor queries.
- **Language Models:** Use freely available LLMs. For example, **Llama-2** (Meta, open commercial license) [6] in 7B or 13B size for a good balance of quality and speed. Other options include Falcon-7B, MPT-7B/Instruct, or Pythia. Hugging Face has many open LLMs fine-tuned for chat (see list of open models [7]). These run locally under permissive licenses.
- **Fine-tuning:** Use Hugging Face Transformers and PEFT/LoRA to efficiently adapt the LLM to the travel domain Q&A, if resources allow. We can apply QLoRA on a free GPU (e.g. Google Colab) to fine-tune on the travel Q/A data.
- **Pipeline/Framework:** A Python stack. Optionally use LangChain or LlamaIndex for easier RAG integration (both are open-source). Otherwise, manually code: use Hugging Face `transformers` for model inference; `faiss`/`sklearn` for retrieval; and connect them in custom logic.
- **Frontend/Backend:** For zero-budget deployment, we can use **Gradio** or **Streamlit** to build a simple web UI (both are free). Hugging Face Spaces natively supports Gradio/Streamlit apps [8]. Alternatively, a Flask or FastAPI backend (free tier) with a minimal JavaScript/HTML UI could be used.
- **Deployment:** We plan to deploy on Hugging Face Spaces (free tier) or a free cloud app service. Spaces allow hosting a Gradio chat interface with our model for free [8]. We will containerize the app (Spaces supports Docker) if needed, but simplest is a Python Space.
- **Additional Tools:** Git/GitHub (or HF Hub) for version control, Hugging Face Datasets library for data, Python (PyTorch) environment.

All selected tools and models are open-source or have free usage tiers. No paid APIs (like OpenAI) will be used.

5. Development Plan

1. **Data Ingestion & Cleaning:**
2. Acquire the Bitext travel QA dataset and any other open travel corpora. Use `datasets` or `requests` to download.

3. Extract text content from guides/FAQs (e.g. using `BeautifulSoup` if scraping public webpages like Wikipedia).
4. Clean and normalize text: remove HTML tags, anonymize any sensitive tokens (account IDs, etc.), and remove duplicates.
5. Split data into training/validation for any fine-tuning.
6. **Knowledge Base Construction & Indexing:**
7. **Assemble Documents:** Combine cleaned texts into knowledge-base documents. This might include: (a) Q&A pairs from the dataset (we could treat Q+A together as one doc); (b) travel help articles (trip planning, baggage rules, ticketing guides); (c) travel FAQs from open sites.
8. **Chunking:** Break long documents into smaller passages (~150 words) to improve retrieval relevance.
9. **Embedding:** Use a pre-trained Sentence-BERT model (e.g. `all-MiniLM-L6-v2`) to encode each passage into a vector.
10. **Indexing with FAISS:** Build a FAISS index from these vectors. We'll experiment with index types (IVF, HNSW) for speed vs. accuracy.
11. **Test Retrieval:** Run sample queries (e.g. "How do I cancel my flight?") to ensure the FAISS search returns relevant passages. Manually inspect hits.
12. **Generative Model Selection & Tuning:**
13. **Choose Base Model:** Start with Llama-2-7B or Falcon-7B. These are small enough to run on modern GPUs/CPU in reasonable time.
14. **Fine-Tuning (Optional):** Fine-tune the model on travel Q/A. Using the prepared Q&A pairs, apply LoRA/QLoRA with Transformers Trainer. This adapts the model's style and knowledge to the travel domain.
15. **Prompt Engineering:** Develop prompts that include retrieved context. For example:

```
System: You are a travel assistant. Use the information below to answer the
user's question.
[Context: "passage1..."; "passage2..."]
User: {user_query}
Assistant:
```

16. **Validation:** Test the model on held-out questions to check answer quality. Compare answers with/without retrieval to justify the RAG approach.
17. **Dialogue Management & Integration:**
18. **Conversation State:** Decide if multi-turn history is needed. (We can start stateless: each query is independent, or carry last turn embeddings for context.)
19. **RAG Pipeline:** Implement a handler function: (a) embed the user's message; (b) query FAISS for top-k docs; (c) format prompt with docs + query; (d) run LLM to generate answer; (e) return answer.

20. **Fallbacks:** If the question is off-domain or retrieval fails, the LLM can still attempt a safe answer (or say "I don't know"). Optionally include a FAQ fallback for common cases (e.g. a keyword match before LLM).

21. **Testing:** Iteratively test with sample conversations. Ensure relevance and correctness of answers. Tune number of retrieved docs and LLM parameters (temperature, etc.).

22. **Frontend/Backend Setup:**

23. **Prototype UI:** Create a simple chat interface. For example, use **Gradio** to build a web chat widget that calls our pipeline on submit. Gradio integrates easily with Python code.
24. **Backend API:** If not using Spaces, set up a Flask/FastAPI app with an endpoint `/chat` that accepts a user message and returns the bot's reply. The endpoint runs the retrieval+generation code.
25. **Packaging:** Containerize the app (using Docker) or prepare it for deployment on Hugging Face Spaces. Include all necessary files (model weights, index, code).
26. **Iteration:** Deploy an initial version, test end-to-end with real queries, debug issues (time-outs, token limits, UI bugs).

The development follows an agile approach: we'll continuously test each part before moving on, ensuring the pipeline is robust.

6. Deployment Strategy

We recommend **Hugging Face Spaces** for zero-cost deployment. Spaces allow free hosting of ML apps with Gradio or Streamlit front-ends [8]. We can push our code to a new Space (select "Gradio" environment), and it will automatically build and serve the app. Spaces even provides CPU compute for inference (GPU is available via paid plan if needed).

Steps: - Upload the fine-tuned model and FAISS index to Hugging Face Hub (as a private or public repo).
- In a new Space, install required packages (transformers, faiss-cpu, sentence-transformers).
- Use Gradio blocks to create a chat interface; behind the scenes call the retrieval/LLM pipeline.
- (Alternative) If full-stack web app desired, we could deploy a Flask backend on a free tier (e.g. Render.com or Heroku free tier) and a simple HTML/JS front-end, but Spaces is simpler and integrates directly with our Python code.

By hosting on Spaces we get "free hosting for ML-powered demos" [8], and we can share a URL to end-users.

7. Future Roadmap – Adding Voice Support

Once the text chatbot is stable, we can extend it to voice with these modules:

- **Speech Recognition (ASR):** Use an open-source model like **OpenAI's Whisper**. Whisper (released as open-source) provides robust transcription (multi-language, noisy environments) [9]. We can integrate Whisper by capturing microphone input in the web UI (e.g. via WebRTC) or mobile app, sending audio to the backend, and converting it to text.

- **Text-to-Speech (TTS):** Use an open-source TTS engine. For example, Coqui TTS (MPL-2.0 license) has pretrained models for many languages [10]. The chatbot's text reply can be fed into Coqui to generate audio output.
- **Pipeline Integration:** The dialogue flow becomes: **User speaks → ASR converts to text → existing chatbot answers → TTS vocalizes reply**. We must handle audio I/O in the front-end (HTML `<audio>` or WebSpeech API) and ensure low latency.
- **Considerations:** Initially support one language (English) to simplify. Ensure fallback to text for unsupported languages. Also handle "wake-word" or push-to-talk if continuous listening is not desired.

This roadmap allows us to add a voice layer without redesigning the core chatbot. Both Whisper and Coqui are free/open, so no additional licensing costs are needed.

Sources: We leveraged open-source resources: e.g., travel chatbot datasets [1], RAG principles [4], FAISS indexing [5], and open LLMs [6] [7]. We will deploy on Hugging Face Spaces [8] and can extend to voice using Whisper [9] and Coqui TTS [10]. This ensures a **completely free and open-source** solution from data to deployment.

---

[1] bitext/Bitext-travel-llm-chatbot-training-dataset · Datasets at Hugging Face
https://huggingface.co/datasets/bitext/Bitext-travel-llm-chatbot-training-dataset

[2] Andyrasika/Ecommerce_FAQ · Datasets at Hugging Face
https://huggingface.co/datasets/Andyrasika/Ecommerce_FAQ

[3] Artificial Intelligence in Tourism Through Chatbot Support in the Booking Process—An Experimental Investigation
https://www.mdpi.com/2673-5768/6/1/36

[4] What is RAG? - Retrieval-Augmented Generation AI Explained - AWS
https://aws.amazon.com/what-is/retrieval-augmented-generation/

[5] Welcome to Faiss Documentation — Faiss documentation
https://faiss.ai/

[6] [7] Open-Source Text Generation & LLM Ecosystem at Hugging Face
https://huggingface.co/blog/os-llms

[8] Spaces
https://huggingface.co/docs/hub/en/spaces

[9] Introducing Whisper | OpenAI
https://openai.com/index/whisper/

[10] GitHub - coqui-ai/TTS: - a deep learning toolkit for Text-to-Speech, battle-tested in research and production
https://github.com/coqui-ai/TTS