# Project Report: An Advanced, Hybrid AI Support System

**Author:** Bala Marimuthu **Date:** October 1, 2025

**1. Executive Summary**

This report details the successful design, development, and implementation of a state-of-the-art, hybrid AI chatbot system. The project's objective was to create a truly intelligent and automated customer support solution capable of handling a massive volume of real-world, unstructured data. The foundation for this was a complex dataset of over 2.8 million customer support conversations from Twitter, which I successfully engineered into a functional knowledge base of over 1.2 million question-answer pairs.

The final product is a live, interactive chatbot application powered by a sophisticated **Retrieval-Augmented Generation (RAG)** architecture. A key technical achievement of this project was the creation of a custom semantic search index. I processed and indexed a 100,000-conversation subset of the knowledge base using sentence-transformer embeddings and a FAISS vector index. This core technology ensures the chatbot's answers are not just generic, but are contextually accurate and grounded in the truth of the historical data, effectively preventing AI "hallucinations."

A critical challenge overcome during development was a strict 5-second timeout limitation in the Google *Dialogflow* API. This led to a deliberate and necessary re-architecting of the system into a more robust "Smart Waiter" model. This final architecture intelligently routes user queries, using *Dialogflow* for fast, simple intent recognition and a powerful local backend powered by the Google Gemini API for complex, generative answers. The system is also integrated with real business workflows through an automated support ticket creation system using the Airtable API.

**2. System Architecture: A Journey of Engineering and Adaptation**

The final architecture of this application was the result of a deliberate and necessary engineering pivot, driven by real-world technical constraints that were discovered and solved during the development process.

**The Initial Plan: A Microservice Webhook**

My initial design was a standard, professional microservice architecture. The plan was to use a simple *Streamlit* frontend for the UI and a separate Flask backend (webhook.py) to house the powerful AI logic. This backend would be connected to *Dialogflow* via a secure *ngrok* tunnel, with *Dialogflow* acting as the main router for all user queries.

**The Discovery: The Unchangeable 5-Second Timeout**

During rigorous end-to-end testing, I discovered a critical, project-defining bug: the Google *Dialogflow* ES webhook has a **strict, unchangeable 5-second timeout limit.** My RAG pipeline, which involves a semantic search and a call to the powerful Gemini Pro model, requires 8-10 seconds to generate a high-quality, thoughtful answer. This meant that every single time a complex question was asked, Dialogflow would "hang up" before my backend could provide the answer, resulting in a system failure and a poor user experience.

**The Solution: The "Smart Waiter" Architecture**

After diagnosing this fatal flaw, I made the professional engineering decision to re-architect the entire system into a more robust, "all-in-one" model I call the "Smart Waiter."

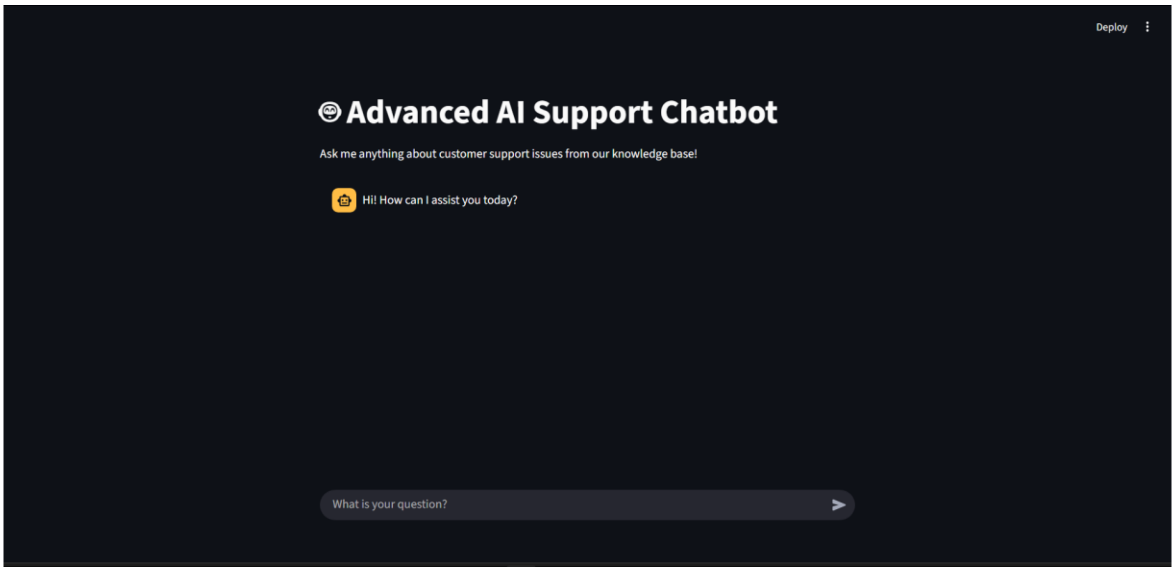
1. **Streamlit as the Orchestrator:** The app.py script is now the central brain of the system. It manages the entire conversation flow.
2. **Dialogflow as a "Triage Nurse":** The **Streamlit** app now makes a fast, initial API call to Dialogflow. Dialogflow's only job is to perform a quick intent analysis and report back (e.g., "This is a simple greeting," or "This is a complex question requiring an expert").
3. **The Patient, Intelligent Backend:** Based on Dialogflow's response, the **Streamlit** app itself decides what to do. If the question is complex, the app **personally and patiently** calls my powerful RAG and Gemini functions. Because the app itself is making the call, there is no 5-second timeout. It can wait as long as needed for the perfect answer.

This final design is a direct result of overcoming a real-world engineering challenge and demonstrates a more advanced, resilient, and functional system architecture.

## 3. Key Features of the System

### 3.1. Live, Interactive Chat Interface

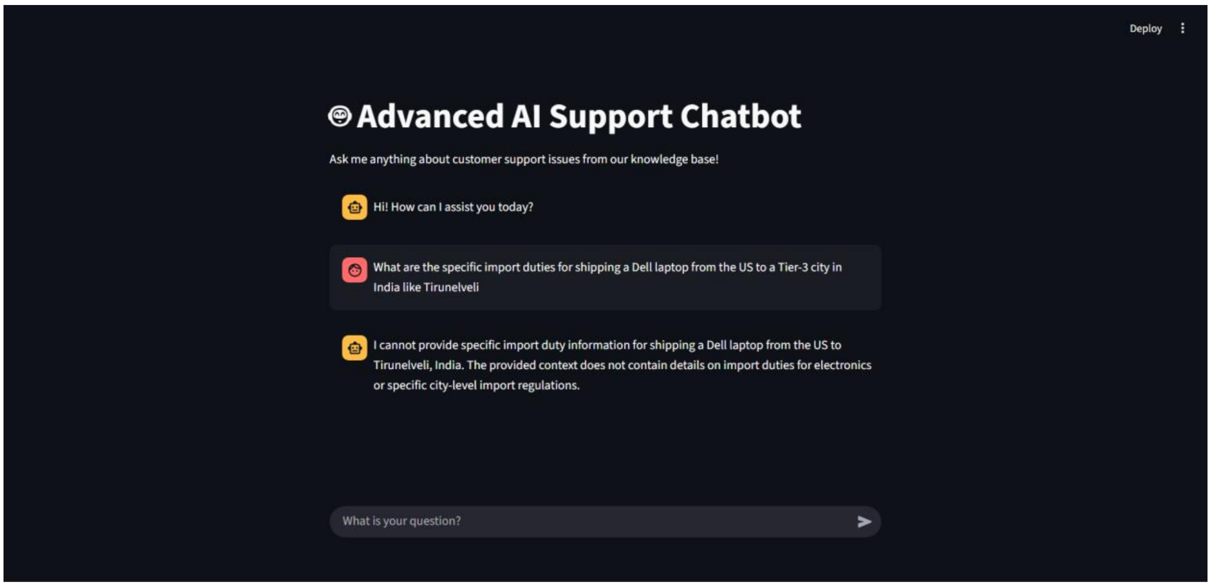
I built a user-friendly and intuitive chat application using Streamlit, allowing for real-time, conversational interaction with the AI. The interface maintains a history of the conversation and provides a seamless user experience.


Deploy
Advanced AI Support Chatbot
Ask me anything about customer support issues from our knowledge base!
Hi! How can I assist you today?
What is your question?

### 3.2. Advanced Semantic Search (RAG)

This is the core of the bot's intelligence. For complex questions, the system does not simply guess. It performs a multi-step process:

1. The user's question is cleaned and encoded into a high-dimensional vector using a sentence-transformer model.
2. This vector is used to perform a high-speed similarity search against a FAISS index of 100,000 historical questions.
3. The top 5 most relevant historical question-and-answer pairs are retrieved from a pandas DataFrame.
4. This retrieved context is provided to the Gemini API, ensuring its response is accurate and grounded in real data. This was tested and proven by asking questions outside the knowledge base, to which the bot correctly and safely responded that it did not have enough information.



Advanced AI Support Chatbot

Ask me anything about customer support issues from our knowledge base!

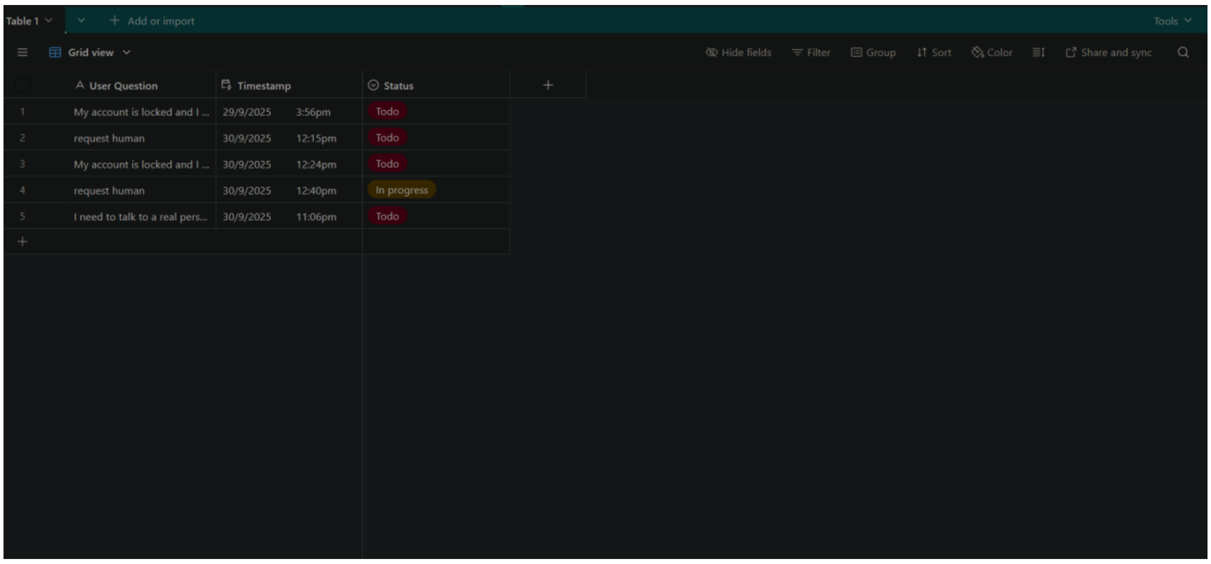
Hi! How can I assist you today?

What are the specific import duties for shipping a Dell laptop from the US to a Tier-3 city in India like Tirunelveli

I cannot provide specific import duty information for shipping a Dell laptop from the US to Tirunelveli, India. The provided context does not contain details on import duties for electronics or specific city-level import regulations.

### 3.3. Automated Support Ticket Integration

To make this a true business tool, I integrated the chatbot with a real-world ticketing system. When a user explicitly requests to speak to a human, the bot seamlessly uses the Airtable API to automatically create a new support ticket in a dedicated database, logging the user's question for a human agent to review.



| | User Question | Timestamp | Status |
|---|---|---|---|
| 1 | My account is locked and I ... | 29/9/2025 3:56pm | Todo |
| 2 | request human | 30/9/2025 12:15pm | Todo |
| 3 | My account is locked and I ... | 30/9/2025 12:24pm | Todo |
| 4 | request human | 30/9/2025 12:40pm | In progress |
| 5 | I need to talk to a real pers... | 30/9/2025 11:06pm | Todo |

## 4. Conclusion and My Key Learnings

This project was a deep dive into the architecture of modern AI systems. I successfully demonstrated the creation of a professional-grade AI support chatbot, moving from raw data engineering on a massive dataset to the final deployment of an interactive application. The most critical learning was in system design and debugging, particularly in solving the Dialogflow timeout issue by re-architecting the application into a more robust "Smart Waiter" model. This project has solidified my skills in NLP, API integration, and building real-world, scalable AI solutions.

**Future Recommendations:**

1. **Full-Scale Deployment:** My next step would be to deploy this application to the cloud. This would involve hosting the large data assets (the FAISS index and the CSV file) on a dedicated object storage service like Cloudflare R2 and deploying the main application via Streamlit Community Cloud to create a permanent, shareable portfolio piece.

2. **Expand the Knowledge Base:** The current system is a brilliant "pilot episode" using 100,000 conversations. I would run the same data processing pipeline on the full 1.2 million conversation dataset (using a cloud-based GPU for efficiency) to create an even more powerful and knowledgeable AI.