Production-Grade RAG Chatbot for JioPay Customer Support

LLM Assignment 2

Swaroop Vaze (2022300134)
Aadi Shetty (2022300113)

September 20, 2025

Contents

1 Abstract

This report details the design, implementation, and evaluation of a production-grade Retrieval-Augmented Generation (RAG) chatbot developed to automate customer support for JioPay. The system leverages publicly available data from the JioPay business website and help center to provide accurate, context-aware, and verifiable answers to user queries. We constructed a robust data ingestion and retrieval pipeline using Python, Flask, ChromaDB, and the OpenAI API. To optimize performance and quality, we conducted a series of ablation studies on key system components, including chunking strategies, embedding models, and retrieval parameters (top-k). The final deployed system utilizes a configuration of 512-token chunks with a 64-token overlap, OpenAI's `text-embedding-3-small` model, and retrieves the top 5 most relevant documents. This configuration achieves a 100% success rate on our test set, with excellent answer quality and an average query latency of approximately 2 seconds.

2 System Overview

The chatbot is built on a classic RAG architecture designed to ensure that all generated answers are grounded in a verified knowledge base, thereby minimizing hallucinations and providing citations. The system is composed of an offline ingestion pipeline and an online inference pipeline.

2.1 System Architecture

1. **Data Ingestion (Offline):** Publicly accessible content from the JioPay website and help center is scraped, cleaned, and segmented into smaller text chunks.
2. **Indexing (Offline):** Each text chunk is converted into a high-dimensional vector embedding using the `text-embedding-3-small` model. These embeddings are then stored and indexed in a ChromaDB vector database for efficient similarity search.
3. **Retrieval (Online):** When a user submits a query, it is first converted into an embedding using the same model. The system then queries the ChromaDB instance to retrieve the top-k most semantically similar text chunks from the knowledge base.
4. **Generation (Online):** The retrieved chunks are compiled into a context block and prepended to the user's original query. This combined text is then passed to a generator LLM (`gpt-3.5-turbo`), which synthesizes a final answer based only on the provided context, along with citations pointing to the source documents.

The backend is implemented as a Flask web server, providing a REST API for the frontend to interact with the RAG pipeline.

3 Data Collection

The knowledge base was constructed in strict accordance with the assignment guidelines, using only publicly accessible information to ensure compliance and ethical data handling.

3.1 Data Sources

- **JioPay Business Website:** All relevant pages under `jiopay.com/business` were programmatically crawled and their textual content was extracted.
- **JioPay Help Center / FAQs:** All question-and-answer pairs and help articles were scraped to cover common customer support scenarios.

3.2 Compliance

The scraping process was configured to respect the `robots.txt` file of the source domains. No gated content, personal user data, or copyrighted material outside the scope of public help documentation was accessed. The collected data was stored locally for the ingestion process.

4 Ablation Studies: Component Optimization

To determine the optimal configuration for the RAG pipeline, we conducted systematic ablation studies on three critical components: chunking strategy, embedding model, and the number of retrieved documents (top-k). The primary evaluation metrics were average retrieval score (cosine similarity), average answer quality (on a binary scale of 0 to 1, assessed by humans), success rate (%), and P95 latency (ms).

4.1 Chunking Ablation

The goal of this study was to identify the most effective method for segmenting documents. We compared a baseline ”structural” chunking method against recursive character splitting with varying chunk sizes and overlaps.

Table 1: Chunking Strategy Comparison

Strategy	Size	Overlap	Avg Retrieval Score	Avg Answer Quality	P95 Latency (ms)
structural	-	-	0.5981	1.0	3035.53
recursive	256	32	0.5281	1.0	2588.62
recursive	**512**	**64**	**0.5981**	**1.0**	**2586.90**
recursive	1024	128	0.5981	1.0	3254.40

Insights: While all tested configurations achieved perfect answer quality, the recursive strategy with a size of 512 and overlap of 64 provided the best balance. It matched the highest retrieval score of the structural method while demonstrating significantly lower P95 latency. The smaller chunk size (256) hurt retrieval performance, while the larger size (1024) increased latency without providing any quality benefit. This justifies its selection for the final system.

4.2 Embeddings Ablation

This study aimed to select the best embedding model, balancing performance, cost, and quality.

Table 2: Embedding Model Comparison

Provider	Model	Avg Retrieval Score	Avg Answer Quality	P95 Latency
Local	`all-MiniLM-L6-v2`	0.5281	1.0	1235.15
Cohere	`embed-english-v3.0`	0.4533	1.0	2840.85
OpenAI	`text-embedding-3-small`	**0.5981**	**1.0**	**2749.31**
OpenAI	`text-embedding-3-large`	0.5981	1.0	3821.11

Insights: OpenAI’s `text-embedding-3-small` emerged as the clear winner. It delivered the highest retrieval score, matching its much larger and higher-latency counterpart. While the local model offered the lowest latency, its retrieval performance was significantly weaker. The `text-embedding-3-small` model provides state-of-the-art performance at a reasonable latency and cost, making it the optimal choice.

4.3 Retrieval Ablation (Top-k)

This experiment was conducted to determine the optimal number of documents (k) to retrieve and pass to the LLM.

Table 3: Top-k Retrieval Comparison

Top-k	Avg Retrieval Score	Avg Tokens	Avg Answer Quality	P95 Latency (ms)
3	0.5982	639.1	1.0	2561.19
5	**0.5981**	**929.4**	**1.0**	**2586.90**
7	0.5981	1220.0	1.0	2822.23
10	0.4736	1666.9	1.0	2401.53

Insights: The results show that retrieval scores peak between $k = 3$ and $k = 7$. Increasing k to 10 significantly degraded the average retrieval score, suggesting that the additional documents were noisy and irrelevant. Setting $k = 5$ provides an excellent compromise. It maintains a near-perfect retrieval score while keeping the token count, and thus latency and cost, manageable.

5 Evaluation Summary

The comprehensive ablation studies confirm that our selected system configuration is robust and well-justified. The chosen components work in concert to deliver high-quality results efficiently.

5.1 Final Pipeline Performance

- **Chunking:** Recursive, Size=512, Overlap=64
- **Embedding Model:** `openai/text-embedding-3-small`
- **Retrieval:** Top-k = 5

5.2 Overall Performance

- **Average Latency:** ∼2068 ms
- **P95 Latency:** ∼2587 ms
- **Average Retrieval Score:** ∼0.598
- **Answer Quality & Success Rate:** 100% on the evaluation test set.

The chatbot is well-equipped to handle customer support queries for JioPay with speed, accuracy, and reliability.

6 Deployment

The developed Production-Grade RAG Chatbot for JioPay Customer Support has been successfully deployed and made accessible on Hugging Face. This deployment allows for broader access and demonstration of the system's capabilities.

Live Link: https://huggingface.co/spaces/svaze/jiopay-support-assistant

7 Limitations

It is important to note that there are slight differences between the code available on GitHub and the deployed version on Hugging Face. These variations primarily stem from the handling and usage of API keys, which are necessary for the system's operation but are managed differently in a public deployment environment compared to a local development setup.