Conference Paper Title*

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

In recent years, conversational AI has transformed customer support by providing instant, accurate, and scalable responses. Traditional chatbots often rely on predefined rules or static FAQs, limiting their ability to handle complex or unforeseen queries. Retrieval-Augmented Generation (RAG) offers a promising alternative by combining information retrieval with large language models (LLMs), enabling the generation of contextually accurate and citation-backed answers.

This project focuses on building a production-grade RAG chatbot for JioPay, leveraging publicly accessible business and help center data. The system not only retrieves relevant information from the knowledge base but also generates natural language responses while providing transparent citations. Through ablation studies on chunking strategies, embeddings, and ingestion pipelines, this work explores the optimal configurations for performance, accuracy, and efficiency, culminating in a deployable web application accessible via a public URL.

II. Methedology

The development process followed a structured pipeline, beginning with the collection of publicly available JioPay data from the business website and help center. The raw content was cleaned, normalized, and segmented using multiple chunking strategies to enable effective retrieval. Embeddings were generated using different state-of-the-art models and stored in a vector database to support dense similarity search. A retriever module was designed to fetch the most relevant chunks, which were then passed to a large language model for grounded response generation with inline citations. Finally, a user-friendly web interface was built to deliver responses interactively, while ablation studies were conducted to evaluate the impact of chunking, embeddings, and ingestion pipelines on performance and accuracy.

A. System Overview

The proposed system implements a production-grade Retrieval-Augmented Generation (RAG) pipeline to automate customer support for JioPay. The workflow begins with data ingestion, where Playwright was used to scrape dynamic content from the JioPay business website and help center. The extracted data was then cleaned and structured into a normalized format.

For knowledge base construction, a Structural-Hierarchical chunking strategy was adopted to preserve contextual hierarchy from headings and sub-sections, ensuring that responses remain grounded in the document structure. These chunks were transformed into dense vector representations using Voyage embeddings (MongoDB), which balance semantic quality with indexing efficiency. The embeddings were stored in a vector database for high-performance retrieval.

The retrieval module performs top-k similarity searches to fetch the most relevant context. These retrieved chunks are passed into Google Gemini 2.0, which generates natural language responses while anchoring outputs to the supporting sources. Inline citations referencing original URLs are included to enhance transparency and reduce hallucinations.

On the deployment side, the system is divided into two layers: a backend implemented in Python to handle ingestion, embedding, retrieval, and LLM integration, and a frontend developed using TypeScript + React to provide an interactive chat interface. The frontend is hosted on Vercel, while the backend is deployed on Render, ensuring scalability, availability, and a clean separation of concerns.

Together, this architecture enables reliable, accurate, and explainable customer support responses, backed by systematic evaluation through ablation studies.

B. Data Collection

To construct the knowledge base for the RAG system, publicly available content from the JioPay Business website and Help Center/FAQs was collected using a combination of Playwright and Puppeteer. Both tools were used on the same set of links to maximize coverage and reliability.

- Playwright: Proved to be more robust and easier to implement. It successfully extracted the majority of textual content and was able to capture associated images as well. However, it did not retrieve all PDF documents, requiring additional handling.
- Puppeteer: Faced some challenges with webpage loading, often requiring multiple runs to complete scraping. Despite this, it managed to capture nearly all PDFs, giving it an advantage in document completeness, though with slightly higher operational overhead.

Additionally, the FAQ section posed difficulties since its content could not be directly scraped with standard methods. A custom script had to be written to extract the FAQ data effectively. PDF resources linked on the site were also downloaded separately to ensure comprehensive coverage of all relevant materials.

Together, the dual-scraping approach combined the strengths of Playwright and Puppeteer, yielding a cleaner and more complete dataset than either could achieve independently.

C. Chunking Ablation

Chunking Ablation

Chunking defines how the raw text is segmented before embedding and indexing. Since different strategies can influence retrieval quality and efficiency in distinct ways, multiple approaches were implemented for comparison.

1) Fixed Chunking The text was divided into fixed-length segments, independent of semantics or structure. Three settings were tested:
   - 256 tokens, 0 overlap → Creates small, non-overlapping segments for fine-grained retrieval.
   - 512 tokens, 64 overlap → Uses moderate-length chunks with slight overlap to retain continuity.
   - 1024 tokens, 128 overlap → Produces larger segments with more context at the cost of index size.

   This method provides predictable segment sizes and straightforward indexing.
2) LLM-Based Chunking Instruction-aware segmentation was performed using a large language model, with each chunk capped at a maximum of 256 tokens. This approach produces variable chunk sizes depending on the LLM's interpretation of semantic boundaries.
3) Recursive Chunking A fallback approach was applied where text was initially divided into large units and then recursively split into smaller units if the size exceeded thresholds. Three configurations were tested:
   - Large → Prioritizes fewer, broader segments before fallback splitting.
   - Balanced → Uses a mid-level segmentation with gradual fallback.
   - Small → Generates fine-grained segments by recursively breaking down text to smaller units.

   This ensured no text block exceeded the target size while retaining coverage of the full document.
4) Semantic Chunking Segmentation was guided by sentence- and paragraph-level embeddings, merging based on similarity thresholds. Three threshold settings were applied:
   - High similarity → Merges sentences/paragraphs only when strongly related.
   - Medium similarity → Balances semantic closeness with coverage.
   - Low similarity → Groups loosely related text into larger units for broader context.

   This resulted in variable-length chunks shaped by semantic coherence.

   5. Structural Chunking Segmentation was aligned with the document hierarchy, preserving boundaries such as headings, sub-sections, and HTML tags. Three settings were applied:
   - Balanced → Splits by headings and sub-sections while maintaining moderate chunk size.
   - Hierarchical → Preserves full document hierarchy, aligning chunks with nested structures.
   - Large → Produces extended segments by combining higher-level structural blocks.

   This method followed the structural divisions of the source documents, resulting in semantically grouped units of text.

D. Emebdding Ablation

Embeddings convert textual chunks into dense vector representations, enabling semantic similarity search in the RAG pipeline. To evaluate the effect of embedding model choice on retrieval and generation, three models were selected and tested with two different chunking strategies: Structural-Hierarchical and Structural-Balanced.

1) Google Embedding GEMMA 300M This proprietary embedding model produces vectors of 768 dimensions. It is optimized for semantic similarity tasks and provides efficient performance in commercial use cases. It was used with both selected chunking strategies to evaluate its compatibility with hierarchical and balanced segmentation.
2) Qwen3-0.6B An open-source embedding model with 1024-dimensional vectors, Qwen3-0.6B is designed to capture richer semantic representations. Its higher-dimensional outputs allow for potentially finer-grained similarity detection between chunks, though at the cost of increased storage and computation. Both chunking methods were paired with this model to observe its behavior on varying chunk structures.
3) Voyage by MongoDB Voyage is an proprietary embedding model also producing 1024-dimensional vectors, with a focus on scalable vector storage and retrieval in document-heavy environments. It was tested with both Structural-Hierarchical and Structural-Balanced chunking to analyze retrieval quality and integration with MongoDB's vector infrastructure.

The primary objectives of this ablation study were:

- Open-source vs Proprietary → To compare licensing, ease of integration, and retrieval performance between models.
- Embedding dimensionality → To evaluate whether higher-dimensional embeddings provide improved semantic representation and retrieval accuracy compared to smaller vectors.

By systematically combining chunking strategies with these three embedding models, the study aims to characterize the trade-offs in retrieval fidelity, index size, and computation cost,

providing guidance for selecting embeddings in production RAG systems.

E. Retrieval + Generation

The retrieval and generation pipeline is central to the RAG system, enabling user queries to be answered accurately and transparently using the structured knowledge base. It is designed to separate embedding, retrieval, and generation components for modularity, scalability, and evaluation.

1) Query Embedding User queries are first transformed into dense vector representations using the Voyage embedding model. The model processes the query text and produces a high-dimensional embedding that captures semantic meaning. These embeddings allow the system to perform similarity-based search over the knowledge base, independent of exact keyword matches.
2) Vector Search and Retrieval The query embedding is submitted to a Pinecone vector index, which contains embeddings for all document chunks generated from the data ingestion and chunking stages. A top-k retrieval is performed to select the most relevant chunks. Each retrieved chunk includes metadata such as source title, source URL, and a text snippet. This allows the system to provide context-aware responses while preserving traceability of information.
3) Context Construction The retrieved chunks are formatted into a structured context for the LLM. Each chunk is enumerated and includes its similarity score, source title, and text content. Parallelly, a list of citation objects is created, which includes the chunk text (truncated for display), source title, and URL. This ensures that the system can provide both the answer and referenceable sources for transparency.
4) Prompt Engineering for Generation The structured context is combined with a prompt instructing Google Gemini 2.0 to generate responses strictly based on the retrieved context. The prompt specifies the desired tone (professional and helpful), output format, and instructions to include citation markers. The generation parameters, including temperature and maximum output tokens, are configured to balance informativeness with conciseness.
5) Response Assembly The model's output is processed to produce the final answer text. The structured citation list is also returned, allowing the frontend to display retrieved sources alongside the answer. This approach ensures that responses are grounded in the knowledge base and can be traced back to original documents.
6) API Integration The retrieval and generation pipeline is exposed via a FastAPI backend. It includes endpoints for:
   /api/chat → Processes user queries and returns responses with citations.
   /api/stats → Provides statistics on the vector index, such as total vectors, dimensions, and index fullness.
   /api/test-retrieve → Returns raw retrieval results for testing and debugging.
7) Logging and Monitoring Comprehensive logging is implemented to track query execution time, retrieved chunk metadata, and response generation. This allows monitoring of latency, debugging of retrieval anomalies, and analysis of system performance during ablation studies.

This design enables modular experimentation with different chunking strategies, embeddings, and generative models while maintaining a robust, production-ready architecture suitable for deployment.

III. Results

Following the development of the RAG system, including data ingestion, chunking, embedding, and retrieval-generation components, the next step involves evaluating the system's performance. The evaluation focuses on the effectiveness of different chunking and embedding strategies, the quality of retrieved context, and the fidelity of generated responses. The results presented below summarize the outcomes of these experiments, highlighting system behavior across multiple configurations.

TABLE I
TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

Fig. 1. Example of a figure caption.

IV. Conclusion

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[4] K. Elissa, "Title of paper if known," unpublished.
[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.
[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

APPENDIX