**Final Project Report**

**Consumer Lens: “Empowering Consumer Protection”**

**Professor: Ryan Wesslen**

**Team members**

**Vishal Sai Devisetty**

**Koushik Meesala**

**Addula Jahnavi**

**1. Executive Summary and Product Overview**

Consumer Lens is built with the mission to revolutionize consumer complaint analysis by simplifying data exploration and empowering stakeholders to drive meaningful change. By leveraging advanced Retrieval-Augmented Generation (RAG) technology, Consumer Lens allows stakeholders to unlock valuable insights from complaint datasets, enabling better decision-making, improved consumer outcomes, and enhanced accountability.

At its core, Consumer Lens is about empowering stakeholders to not only analyse but also understand and act on consumer complaints efficiently and effectively. Its key value proposition lies in its ability to promote transparency and actionable insights, making it an essential tool for organizations prioritizing consumer trust and satisfaction.

**key features and capabilities**

Consumer Lens simplifies consumer complaint analysis through an intuitive queryable interface, enabling seamless dataset integration. It uncovers trends and actionable patterns, delivering precise insights to help stakeholders resolve complaints efficiently. By fostering transparency and accountability, Consumer Lens supports ethical decision-making and empowers businesses, advocates, and regulators to enhance consumer trust and satisfaction.

**Target Users:**

**Consumer Advocates:** Utilize Consumer Lens to identify systemic consumer issues and advocate for meaningful changes effectively.

**Regulatory Bodies:** Monitor compliance with consumer protection laws and ensure fair treatment across industries.

**Businesses:** Enhance complaint management systems, resolve issues efficiently, and foster stronger customer relationships.

**Policy Makers:** Analyse complaint trends to make informed, data-driven decisions for impactful policy reforms.

**Use Cases:**

Identifying the most common consumer grievances across various sectors.

Tracking complaint resolution timelines to improve response efficiency and quality.

Detecting emerging trends in consumer dissatisfaction to shape proactive strategies and solutions.

**Discussion of Ethical Considerations and Mitigations**

Consumer Lens is designed with a strong emphasis on ethical considerations to ensure responsible and secure usage. It anonymizes personal information and safeguards sensitive data to maintain privacy. The platform minimizes biases in data interpretation through a fine-tuned RAG model, promoting fair and equitable insights. Transparent methodologies accompany all analyses, fostering trust and clarity among users. By encouraging action-oriented resolutions over punitive measures, Consumer Lens supports constructive outcomes. Additionally, its user-friendly design ensures accessibility for stakeholders with varying levels of technical expertise, promoting inclusivity and widespread usability.

**2. Technical Implementation**

**Model Selection and DeploymentStrategy**

The system utilizes Meta-Llama-3-8B-Instruct, a fine-tuned version of Llama 3 tailored for instruction-following tasks. This model was chosen for its balance of performance and computational efficiency, making it ideal for delivering clear, task-specific responses in conversational AI systems.

The model is hosted on the Modal platform, leveraging serverless infrastructure to ensure cost efficiency, scalability, and ease of deployment. Resources scale dynamically to meet traffic demands, optimizing performance while minimizing expenses.

The model is deployed as a web service via an API, enabling seamless interaction between the frontend and backend. This API-driven architecture allows for flexible frontend integration and reusability across applications.

The Streamlit framework serves as the frontend, offering an intuitive interface for user interaction. User queries are sent to the Modal API, processed by the model, and displayed back to users in the Streamlit application.

**RAG Implementation Details and Data Sources**

The system utilizes Retrieval-Augmented Generation (RAG) to enhance response quality by combining generative AI with context retrieval from a vector database. Weaviate serves as the vector database, storing embeddings of preprocessed document chunks under a schema named Credit Complaints. The embeddings are generated using OpenAI's text-embedding-3-small model, enabling high-dimensional vector representation of textual data. When a user submits a query, it is converted into a vector embedding, and a similarity search retrieves the most relevant context from Weaviate. The retrieved context is then concatenated with the query to create an augmented prompt, which is processed by a GPT-based model which is GPT-4 for generating responses.

The data source primarily includes structured datasets of customer complaints and resolutions, categorized by financial institutions and issues. The data is preprocessed into manageable chunks for efficient embedding and retrieval. The system evaluates its performance using DeepEval, employing metrics like Correctness (GEval) for factual accuracy, Faithfulness to assess alignment with retrieved context, and Contextual Relevancy to measure relevance. The integration of Streamlit for the frontend ensures a seamless user interface for query submission and response display. This RAG setup effectively combines retrieval and generation, ensuring that responses are accurate, contextually relevant, and factually grounded.

**Integration Patterns**

The system leverages a combination of tools and platforms for seamless interaction and efficient processing. Streamlit provides the frontend for user queries, ensuring an interactive and user-friendly interface. The backend relies on Modal for deploying models and managing computational resources dynamically. Weaviate, a vector database, is integrated for storing and retrieving embeddings generated via OpenAI models. The architecture also utilizes Fast API for managing API endpoints, connecting the frontend and backend efficiently.

**Data and Knowledge Engineering**

The system's data engineering pipeline involves transforming raw data into a structured format suitable for embedding and retrieval. Knowledge engineering is achieved by integrating a vector database to store embeddings, enabling efficient retrieval of relevant documents or data chunks to support query responses. A combination of OpenAI embeddings and vector-based similarity search ensures that retrieved contexts align with user inputs.

**Document Preprocessing Techniques**

The document preprocessing pipeline includes cleaning, tokenizing, and chunking large documents into manageable pieces. Techniques like text normalization, stop word removal, and deduplication enhance the quality of the data before it is embedded. Preprocessed chunks are embedded using OpenAI's text-embedding-3-small model and stored in Weaviate for retrieval during query processing.

**Vector Database Implementation**

Weaviate is used as the vector database, providing a robust and scalable solution for embedding storage and retrieval. Schema classes are defined to organize stored embeddings, such as the Credit Complaints class. Queries are matched against embeddings using near Vector similarity searches, retrieving the most contextually relevant chunks.

**Chunking Strategies and Embedding Approaches**

Chunking is performed based on predefined token limits to ensure compatibility with embedding models and maintain context integrity. The approach involves dividing documents into smaller, semantically meaningful parts. OpenAI's text-embedding-3-small model is employed to create high-dimensional vector representations of these chunks, which are stored for later retrieval.

**Chain of Thought and Step-by-Step Reasoning Examples**

The system employs step-by-step reasoning through dynamic prompts, guiding the model to provide detailed and logical responses. These prompts encourage the AI to outline intermediate reasoning steps, improving transparency and accuracy.

**System Prompts and Few-Shot Learning Patterns**

System prompts define the assistant's behaviour, such as "You are a helpful assistant." Few-shot learning is implemented by including examples in the prompt to set expectations for the model's responses.

**Prompt Templates and Dynamic Generation Strategies**

Templates are dynamically generated using user inputs and retrieved context. For example, templates combine question text with retrieved chunks to construct a prompt like:

"Based on the following context: {context}, answer the question: {question}."

These strategies collectively enhance the model's ability to provide relevant, context-aware, and logical outputs.

**3. Testing and Evaluation Results**

**User Testing and Automated Testing Results**

The system underwent extensive testing phases to validate functionality and reliability. User testing involved stakeholders interacting with the Streamlit interface, evaluating usability, accuracy, and performance. Automated testing was conducted using DeepEval, which assessed key metrics such as correctness, faithfulness, and contextual relevancy. These evaluations were based on predefined test cases covering various queries and scenarios.

**Correctness (GEval)** score: 33.33%, reflecting opportunities for improvement in aligning model outputs with expected results.

**Faithfulness** score: 66.67%, demonstrating that responses were largely grounded in the retrieved context and factual accuracy.

**Contextual Relevancy** highlighted the need for improved retrieval mechanisms to ensure that retrieved data aligns closely with user queries.

**Examples of Improvements Made Based on Feedback**

User feedback led to key enhancements, including optimizing Weaviate queries for better context retrieval, streamlining the Streamlit interface for improved usability, and refining prompts with detailed templates to enhance clarity and accuracy in responses.

**Performance Benchmarks**

The system demonstrated strong performance benchmarks, with an average latency of 2-3 seconds for processing user inputs and retrieving responses, ensuring a smooth and responsive user experience. Additionally, it achieved an average token throughput of 20-30 tokens per second during response generation, reflecting efficient handling of computational tasks.

**Security and Safety Considerations**

The system prioritizes security and fairness through multiple measures. Input validation and output filtering mechanisms prevent harmful or irrelevant content, ensuring safe user interactions. Dynamic prompt construction and the use of escaped inputs effectively mitigate prompt injection risks, maintaining system integrity. Bias testing evaluated the system's performance across diverse demographic and contextual queries, revealing generally neutral responses. Minor biases identified due to skewed training data were addressed by refining

prompt templates and retrieval strategies, ensuring balanced and unbiased outputs.

**4. Future Directions and Lessons Learned**

**Critical Evaluation of Outcomes**

The system demonstrated notable strengths in interactive query handling and context-aware responses, but certain limitations were identified. For instance, while faithfulness and correctness metrics were satisfactory, contextual relevancy showed room for improvement. The system excelled in its modular design and integration of multiple components, but retrieval mechanisms require fine-tuning to enhance response precision.

**Potential Enhancements and Scalability Considerations**

To scale the system further, integrating advanced retrieval-augmented generation (RAG) techniques, such as hybrid search or dense passage retrieval, could improve contextual relevance. Enhancing model fine-tuning with domain-specific datasets and incorporating more robust vector database solutions can support larger-scale applications. Scalability considerations also involve optimizing latency for high-traffic environments through distributed architecture and caching mechanisms.

**Key Learnings About LLM Application Development**

Developing this application highlighted the importance of balancing model complexity with computational efficiency. Lessons learned include the value of user feedback in refining interfaces and prompts, the need for dynamic workflows to address varied user queries, and the significance of continuous testing to identify and resolve issues. This iterative process underscored the importance of modular design and extensibility in long-term application development.

**Reflection on Practical Challenges and Solutions**

Practical challenges such as managing API rate limits and mitigating biases required innovative solutions. Rate limits were addressed through efficient batching strategies and prioritizing essential queries. Bias issues were mitigated by refining prompts and ensuring a balanced training dataset. The iterative feedback loop between automated metrics and human evaluation played a critical role in resolving these challenges effectively.

**Additional Requirements**

Future developments could include more sophisticated feedback mechanisms, such as real-time analytics dashboards, to monitor system performance and user satisfaction. Expanding the system’s applicability to multilingual settings and incorporating accessibility features for diverse user groups will further enhance its usability and inclusivity.

GitHub link: <https://github.com/Vishaalsai29/operator.git>