Scalable LLM Based Chatbot on Distributed Architecture with RabbitMQ Integration

Anudeep Gadi
Department of Engineering and Computer Science
University of Missouri- Kansas City
aggd4@umkc.edu

Ava Sharif Jourabchi
Department of Engineering and Computer Science
University of Missouri- Kansas City
asddx@umkc.edu

INTRODUCTION

The rise of advanced large language models (LLMs) and embedding techniques has empowered companies to develop FAQ chatbots using internal documents, marking a significant advancement in customer support and knowledge dissemination. However, leveraging external processing services like OpenAI and Voyage AI often necessitates sending confidential data to their servers, raising privacy and security concerns. This can hinder chatbot development for companies with strict data protection policies. Moreover, the creation of embeddings, essential for chatbot training, is both time and resource intensive. Utilizing a single on-premises machine to generate embeddings is not suggested due to its limited scalability and fault tolerance. These challenges underscore the need for innovative solutions that enable on-premises processing of confidential documents while ensuring efficiency and compliance with data security regulations. To address these issues, this paper proposes a novel solution: a distributed architecture with a message broker system, specifically designed for on-premises processing of confidential documents in FAQ chatbot development. This approach eliminates the need for external services, mitigating privacy risks and ensuring compliance with data security regulations. Furthermore, the paper delves into the design and implementation of this system, highlighting its ability to: Process confidential documents on-premises, safeguarding sensitive information and mitigating the risk of privacy breaches. Maintain efficiency and scalability, enabling chatbot development to adapt to growing document volumes without compromising performance and offer fault tolerance, ensuring system reliability and continuous operation even in the face of component failures.

CCS CONCEPTS

Message Broker - AMQP, Remote Procedural Call, Event Notifications, Large Language Model, Retrieval-Augmented Generation

RELATED WORK

The traditional approach to document processing typically involves using a single machine. In this method, a document is segmented into multiple chunks, and embeddings are sequentially generated for each chunk before being stored in a vector database. However, if the embedding model relies on external services like OpenAI or Voyage AI, the data is transmitted to their servers for processing. This raises concerns, particularly with confidential documents, as the data leaves the company's premises [1][2]. Microsoft offers a potential solution to this dilemma through their Azure platform. By setting up an Azure account and utilizing an OpenAI Azure instance, all data processing can be conducted within the company's Azure tenant and the data will be exclusive to the company [3]. However, this solution introduces the challenge of vendor lock-in, limiting flexibility for companies operating on different cloud platforms such as AWS or GCP. Additionally, the rapidly evolving landscape of embedding models, particularly Large Language Models (LLMs), presents a continual challenge. New and improved models are released frequently, raising the question of whether the chosen model remains optimal. To address these issues, a more generic solution is required. Companies seek to conduct all processing tasks using internal resources, but the generation of embeddings poses significant resource and time constraints [4]. At an enterprise level, a scalable, efficient, and fault-tolerant system is essential. It is within this context that the proposed paper seeks to offer a comprehensive solution to the challenges mentioned.

PROPOSED TECHNIQUE

The proposed technique leverages a distributed architecture combined with a message broker system to address the challenge of on-premises processing of confidential documents for FAQ chatbot training.

The technique begins with the segmentation of large confidential documents into smaller chunks. These chunks are then fed into a message broker system, where they are distributed among worker processes responsible for generating embeddings. Once processed, the embeddings are stored in a vector database for later use.

In the next part of answering the queries, we utilize RabbitMQ's Remote Procedure Call (RPC) mechanism. When the server receives queries, they are added to the message queue for processing. A worker then handles the query, providing a response. Additionally, we incorporate a cache feature into this process. Since FAQs often feature repeated questions, we create embeddings of the queries and store them alongside their corresponding answers in another collection. When a query is received, we first check this collection for a reasonable match. If found, we can directly respond with the answer, eliminating the need to involve the Large Language Model (LLM).

Relatedness to distributed computing

Our proposed technique is closely related to distributed computing, as it harnesses the power of distributed architectures and message broker systems to efficiently process large volumes

of data. By distributing the workload among multiple worker processes and utilizing a centralized message broker for communication and coordination, our technique enables parallel processing of document segments, resulting in improved efficiency and scalability.

Aspects of distributed computing used

Event Notification: When a document is uploaded, an event is sent to the message broker, one of the workers takes this event and fetches the document and splits it multiple chunks and adds these chunks to the message queue.
Message Queues: We employ message queues provided by the message broker system to facilitate communication and task distribution between components of our distributed architecture.
Remote Invocation: In the later part when users send their queries to the server, the server does an RPC (flavor provided by RabbitMQ), this is asynchronous. The worker does the processing and sends back the answer.

The solution is horizontally scalable as the number of workers can be increased to handle the workload, and when the load is low, workers can be removed. Additionally, the solution is fault-tolerant; even if one worker fails, other workers can handle the load. Furthermore, with this architecture, document processing occurs in parallel, leading to improved efficiency. By incorporating solutions like Kubernetes, adding, and removing workers becomes straightforward and effortless. This technique represents a formalized and systematic approach to address the complexities associated with on-premises processing of confidential documents, offering improved efficiency, scalability, fault tolerance, and parallel processing capabilities in FAQ chatbot development.

IMPLEMENTATION

Tools/Softwares Used: MinIO (Object Storage), RabbitMQ (Message Broker), ChromaDB (Vector Database), Llama2 (Large Language Model). The solution comprises two main parts: creating embeddings of documents and answering user queries using the stored embeddings.

Creating embeddings of documents:

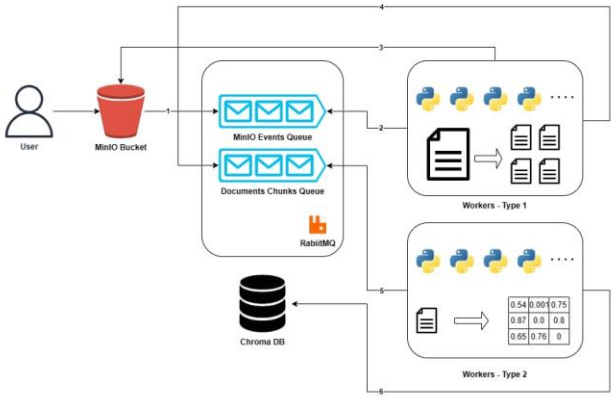

Figure 1: Creation of Embeddings from Document

PSEUDOCODE 1: CREATING EMBEDDINGS

```
ON file upload to MinIO bucket
TRIGGER event to RabbitMQ

EVENT_WORKERS READ events from RabbitMQ's minio_events_queue
FOR EACH event
    PATH = event.file_path
    FILE = MinIO.GET(PATH)
    CHUNKS = FILE.split_into_chunks()
    FOR EACH chunk
        ADD chunk to chunk_queue

EMBEDDING_WORKERS READ chunks from chunk_queue
FOR EACH chunk
    EMBEDDING = CREATE_EMBEDDING(chunk)
    STORE EMBEDDING in ChromaDB
```

The document embedding process begins with the utilization of MinIO, an on-premise object storage system, where documents are stored in buckets similar to Amazon S3. When a document is uploaded to the MinIO bucket, an event is triggered. We have configured MinIO to dispatch these events to a message queue in RabbitMQ. Upon receiving the event, the worker processes read the document's path and retrieves it from MinIO. Subsequently, the document is divided into multiple chunks, and these chunks are enqueued to the message queue for processing. Workers then process these chunks, creating embeddings using an open-source embedding model known as thenlper/gte-large, which operates at a dimensionality of 1024. The generated embeddings are then stored in ChromaDB, a vector database.

Answering user queries

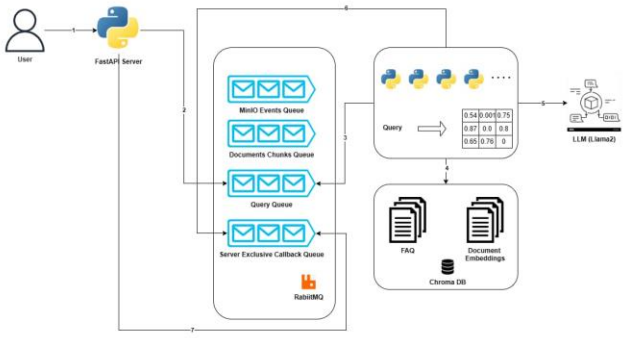

Figure 2: Answering User Queries

The paper's focus lies in retrieving answers from the documents rather than the delivery mechanism to the user. Users have the

flexibility to employ frontend technologies or chatbot frameworks for communication.

PSEUDOCODE 2: PROCESSING QUERIES

```
ON Query Received
    ADD Query to rpc_queue

QUERY_WORKERS READ Queries from rpc_queue
FOR EACH Query
    EMBEDDING = CREATE_EMBEDDING(Query)
    SIMILAR_QUERY =
    SIMILARITY_SEARCH(faq_collection, EMBEDDING,
    top_n = 1)
    IF COSINE_DISTANCE(EMBEDDING,
    SIMILAR_QUERY.embedding) <= 0.1
        SET_ANSWER(SIMILAR_QUERY.answer)
        ADD_ANSWER_TO_CALLBACK_QUEUE(Answer)
        RETURN

    SIMILAR_CHUNK =
    SIMILARITY_SEARCH(documents_collection,
    EMBEDDING, top_n = 1)
    LLM_ANSWER = SEND_TO_LLM(Query,
    SIMILAR_CHUNK)
    SET_ANSWER(LLM_ANSWER)
    SAVE_ANSWER(faq_collection,{Query, LLM,ANSWER})
    ADD_ANSWER_TO_CALLBACK_QUEUE(Answer)

    RETURN Answer to User based on correlation_id
```

Assuming a user sends a query to the server, the server initiates a Remote Procedure Call (RPC) using RabbitMQ. A worker process receives the query, generates its embedding, and conducts a similarity search within ChromaDB to retrieve the most relevant embeddings. Subsequently, the worker retrieves the corresponding page content associated with the relevant embeddings. We employ the Retrieve and Generate (RAG) mechanism, utilizing an LLM (Large Language Model), specifically Llama2, to generate an answer based on the query and the retrieved page content. The generated answer is then sent back to the server, which in turn forwards it to the user.

Additionally, the server creates an embedding of the user query and stores it in a separate collection named FAQ along with the corresponding answer. Upon subsequent queries, the server first checks for similarity within the FAQ collection. If a matching query is found, the corresponding answer is directly returned to the user without involving the LLM. In cases where no similarity is found, the server retrieves similar embeddings from the collections and forwards them to the LLM for further processing.

This implementation effectively addresses the dual objectives of generating embeddings from documents and efficiently responding to user queries while leveraging the capabilities of MinIO, RabbitMQ, and ChromaDB for seamless integration and performance optimization.

EXPERIMENTAL EVALUATION

In order to evaluate the effectiveness of the proposed technique, experiments were conducted to create embeddings of PDF documents of varying lengths. The processing time, measured in seconds, was recorded for different numbers of worker processes.

Experiment 1: 54 pages PDF Document

Number of workers	Time Taken in seconds
1	59
2	39
3	30
4	21

Experiment 2: 102 pages PDF Document

Number of workers	Time Taken in seconds
1	170
2	98
3	73
4	64

The results demonstrate a significant reduction in processing time with an increase in the number of worker processes. The above findings show the scalability and efficiency of the proposed distributed architecture in handling document processing tasks, particularly for larger documents.

CONCLUSION

This research paper introduces a novel approach to tackle the complexities of on-premises document processing for FAQ chatbot development, emphasizing privacy, efficiency, and scalability. By harnessing a distributed architecture and a message broker system, we provide a solution that circumvents the need for external processing services, thereby mitigating privacy concerns and ensuring compliance with data security regulations. Our method optimizes the processing workflow, enabling the creation of embeddings from confidential documents on-site, while maintaining fault tolerance and scalability. Through experimental evaluation, we demonstrated the efficacy of our approach in significantly reducing processing time, particularly evident with larger documents, by dynamically scaling the number of worker processes. Integration of tools like MinIO, RabbitMQ, ChromaDB, and Llama2 ensures a robust and adaptable framework suitable for diverse enterprise environments. This research contributes not only a practical solution to FAQ chatbot development but also sets a precedent for data-sensitive industries, offering a blueprint for leveraging internal resources effectively while upholding stringent privacy standards. Ultimately, our proposed technique represents a pivotal step towards empowering companies with the capabilities to innovate in customer support and knowledge dissemination while safeguarding confidential information.

AUTHOR CONTRIBUTIONS

Anudeep Gadi

Infrastructure Setup: Created a Docker container for RabbitMQ, configured queues, exchanges, and bound them using routing

keys. Managed the setup of MinIO on docker container, creating buckets, and setting up event notifications.
Development: Developed a Python worker script responsible for reading the MinIO events from the message queue, fetching the document and chunking the document and adding them to the message queue. Developed server that is responsible for handling user queries. Developed worker script that generates the answer using LLM and sends to the server using RPC.
Documentation: Designed architecture diagrams to illustrate the system workflow.

Ava Sharif Jourabchi

Research: Conducted research to identify the most suitable embedding models for the project.
Infrastructure Setup: Managed the setup of ChromaDB on docker container to create a robust database infrastructure.
Development: Developed a Python worker script responsible for reading chunks from the message queue and creating embeddings and adding them back to ChromaDB. Developed UI screens for the application.

COLLABORATION STRATEGIES

Utilized a shared spreadsheet to manage tasks, assigning responsibilities and track progress. Employed pair programming for specific functionalities where close collaboration was necessary. Used GitHub for collaborative development and code management.

REFERENCES

[1] Davies, N. 2023. Sharing Your Business's Data with ChatGPT: How Risky Is It? AT&T Cybersecurity Blog. May 22, 2023. https://cybersecurity.att.com/blogs/security-essentials/sharing-your-businesss-data-with-chatgpt-how-risky-is-it

[2] Metomic. 2023. Is ChatGPT a Security Risk To Your Business? & 7 Biggest Chat GPT Security Risks. Metomic Resource Centre. https://metomic.io/resource-centre/is-chatgpt-a-security-risk-to-your-business

[3] Microsoft Learn. 2023. Data, privacy, and security for Azure OpenAI Service - Azure AI services. https://learn.microsoft.com/en-us/legal/cognitive-services/openai/data-privacy

[4] S. K. Routray, A. Javali, K. P. Sharmila, M. K. Jha, M. Pappa and M. Singh, "Large Language Models (LLMs): Hypes and Realities," 2023 International Conference on Computer Science and Emerging Technologies (CSET), Bangalore, India, 2023, pp. 1-6, doi: 0.1109/CSET58993.2023.10346621.

EXTERNAL LINKS

Embeddings creation application –
https://github.com/AnudeepGadi/HeyRoo-chatbot-core

Chatbot and its components -
https://github.com/AnudeepGadi/HeyRoo-chatbot-srv