LLama Learn - Building RAG Flows with LLMs

Ajay Krishnan Gopalan
ag8172
New York University
New York City
ag8172@nyu.edu

Shohna Kanchan
sk11239
New York University
New York City
sk11239@nyu.edu

Devyani Bairagya
db4922
New York University
New York City
db4922@nyu.edu

Dhanesh Baalaji Sreenivasan
ds7636
New York University
New York City
ds7636@nyu.edu

***Abstract*—This paper delves into the construction and hosting of a custom Large Language Model (LLM) and the development of a Retriever-Augmented Generation (RAG) pipeline within the AWS ecosystem. We detail the process of integrating AWS services, including DynamoDB and OpenSearch, to efficiently manage and process extensive datasets. Our findings highlight the effectiveness of this approach in enhancing the performance and scalability of AI-driven language applications in cloud environments.**

***Index Terms*—RAG, LLM, Embedding**

I. Problem Statement

In the modern digital landscape, where vast volumes of textual data are constantly generated, the ability to quickly and accurately extract relevant information from extensive documents presents a formidable challenge. This project aims to address the critical gap in efficient and context-sensitive information retrieval from large-scale textual repositories. Despite the availability of traditional search engines and database systems, they often fall short in understanding the nuanced context of user queries, leading to suboptimal or irrelevant search results. Our proposed solution seeks to leverage the synergy of advanced Natural Language Processing techniques and the prowess of Large Language Models (LLMs) in a Retrieve-and-Generate (RAG) framework. By chunking and vectorizing text content for optimized indexing in OpenSearch, and integrating LLMs for intelligent query processing and response generation, this project aspires to revolutionize the way information is accessed and utilized, providing precise, contextually relevant answers to complex queries. The outcome aims to significantly enhance the efficiency and accuracy of information retrieval, bridging the gap between vast unstructured data sources and the specific informational needs of users across various sectors.

II. Motivation

In an era marked by the exponential growth of digital information, the need for advanced, context-aware information retrieval systems has never been more critical. Our product is designed to address this challenge by harnessing the latest advancements in Natural Language Processing and Large Language Models. By chunking textual content from documents, transforming them into searchable vectors, and indexing them in OpenSearch, our system enables precise and rapid retrieval of information. Coupled with a Retrieve-and-Generate (RAG) framework, it not only fetches the most relevant document fragments in response to user queries but also employs a sophisticated LLM to generate coherent and contextually nuanced answers. This innovative approach not only elevates the efficiency and accuracy of information retrieval but also significantly enhances user experience, making it a transformative solution for various domains grappling with the deluge of digital data.

III. Existing Solutions

RAG models merge two significant components: a retriever and a generator. The retriever fetches relevant documents or data from a large corpus, while the generator uses this information to create coherent and contextually accurate responses. This combination allows the model to produce informative and relevant answers to the input query. Here are some of its existing implementations:

A. Facebook AI's RAG

This model improved in providing more accurate and informative answers compared to traditional generative models and represents a pioneering approach in natural language processing by combining a dense retriever with a sequence-to-sequence generator. This innovative architecture enhances question-answering tasks by efficiently selecting relevant documents or passages from a large corpus of text through dense retrieval techniques and then generating informative, contextually relevant answers using sequence-to-sequence generation. Facebook AI's RAG model has demonstrated improved accuracy and informativeness compared to traditional generative models, making it a valuable tool for various applications where precise, context-aware responses are required. [1]

B. Google's Realm

This model integrates a retrieval system directly into the pre-training process of a language model. This groundbreaking method allows the model to learn not only language understanding but also the ability to retrieve pertinent information from a vast knowledge source as part of its training. By combining these two capabilities, REALM becomes highly effective at answering queries, as it can seamlessly access and incorporate external knowledge. This integration of retrieval and language understanding during training makes REALM a

promising advancement in natural language processing, particularly for applications requiring robust question-answering and information retrieval capabilities. [2]

C. Kensho Nerd

This is a specialized variant of the Retrieval-Augmented Generation model designed for financial and economic contexts. Kensho NERD RAG combines a dense retriever with a sequence-to-sequence generator, enabling it to retrieve and generate information tailored to financial news and data. It excels in recognizing and disambiguating named entities, such as company names or economic terms, within text, making it particularly useful for applications like financial news summarization and event extraction. [3]

D. Khanmigo

Khan Academy introduced Khanmigo, a GPT-4 powered learning guide designed to assist educators and students. Unlike ChatGPT, Khanmigo is a tutor and guide, facilitating more sophisticated and context-aware conversations with students. It helps students find answers by guiding them through problem-solving processes and offering feedback, making it a valuable tool in educational settings. The collaboration between Khan Academy and OpenAI leverages GPT-4's advanced capabilities, and initial testing in various school districts has shown promising results, with plans to expand its usage and refine the system further based on user feedback and demand. [4]

E. Meta's DPR RAG Model

The Dense Passage Retrieval (DPR) model, developed by Facebook AI, is a Retrieval-Augmented Generation (RAG) architecture designed to enhance open-domain question-answering tasks. DPR leverages dense vector representations to retrieve relevant passages from large-scale knowledge bases efficiently. During training, the model learns to encode passages into dense vectors, facilitating fast and accurate retrieval. In the generation phase, a language model is employed to refine and generate responses based on the information retrieved by the dense passages. By incorporating a two-step process involving effective retrieval and subsequent generation, DPR aims to improve the overall performance of open-domain question-answering systems, providing more contextually relevant and accurate answers to user queries. [5]

IV. Architecture

Our RAG system architecture is meticulously designed to prioritize flexibility, scalability, and efficiency. Central to this design is the integration of Lambda functions, microservices, and EC2 instances within EKS clusters. This structure is chosen to bolster customizability and reusability, catering to long-term adaptability and innovation.

The architecture's standout feature is its modular approach. By utilizing Containerizaion, we ensure that incorporating new text embedding techniques or experimenting with different LLM models is a seamless process. Users can easily substitute our default Docker image with their custom versions, adhering to a predefined operational sequence, thereby fostering an environment of continuous innovation.

Lambda functions form the backbone of our system, each meticulously crafted to perform singular tasks or to efficiently manage and assimilate outputs from various microservices, such as AWS Textract. This decoupling not only enhances system resilience by reducing interdependencies but also paves the way for these functions to evolve into independent modules. In the future, these modules can be effortlessly deployed, initialized, and configured upon installation, streamlining the setup process significantly.

The utilization of EKS clusters further accentuates our system's robustness, providing the necessary computational resources while ensuring high availability and effective load balancing. This strategic combination of technologies and practices positions our RAG system as a paragon of modern, scalable, and adaptable NLP architecture, ready to meet both current and future demands with unparalleled efficiency.

Our architecture can be divided into 4 subsections.

A. User Layer

- **Login Process** The user experience commences with a straightforward login mechanism. Users are required to input their email and password, followed by the click of the "LogIn" button. This initial step ensures a secure and personalized entry into the platform.
- **"My Questions and Answers" Page** Upon successful authentication, users are guided to the "My Questions and Answers" page. This meticulously curates and displays all previously posed questions by the user. The page remains empty if the user has not raised questions, preserving a minimalist and organized interface.
- **Exploring Specific Questions** For a more detailed exploration of a particular question, users can click on it. This action triggers the system to reveal the corresponding answer, offering users a closer look into the specifics of their inquiries.
- **Introducing New Content** To introduce fresh content for inquiry, users can navigate to the "Upload Page". This transition guides users to a specialized upload interface, where they can initiate the process by clicking "Upload" Here, they can select a PDF or image and submit it.
- **Questioning the Uploaded Content** Following the successful upload, a text area appears, allowing users to frame specific questions related to the newly uploaded content. This process encourages user engagement and directly connects the uploaded material and the user's questions.
- **Seamless Redirection** Upon submitting their questions, users are redirected to the "My Questions and Answers" page. This allows users to access the platform's response to their inquiry promptly.

Overall, this application helps users quickly get the answers they're looking for and encourages them to actively participate by asking new questions. This approach significantly creates

a positive and productive interaction between the user and the platform.

B. API Layer

Our API layer consists of our API gateway and efficiently manages user authentication, data retrieval, and storage operations, providing a secure and streamlined interaction between clients and our backend services. The integration of AWS Cognito and DynamoDB enhances our system's overall reliability and scalability. It has a total of 3 API endpoints:

- **POST *`/login`*** This endpoint is designed to handle user authentication. When a client sends a request to log in, their email address and password will be provided in the request body. The API gateway triggers a Lambda function responsible for interacting with AWS Cognito to validate the user's credentials. If the credentials are valid, the API returns a status code of 200; otherwise, it returns a status code 401.
- **GET *`/qna`* and POST *`/qna`*** GET /qna fetches a list of questions and answers associated with a specific user. The user ID is provided as a query string parameter. The API gateway communicates with a Lambda function, which queries data from DynamoDB. The response includes a structured JSON format containing the user's Q&A data.
- **POST *`/qna`*** allows users to submit new queries and associated data for storage in the database. The user provides their unique ID, the document ID, and the question string in the request body. The API gateway processes this information and stores it in the database for future retrieval.
- **GET *`/upload/object`*** This endpoint obtains a PreSign URL for securely uploading a document. The document's name is provided as a path parameter. The API gateway generates and returns a PreSign URL, allowing clients to upload the document directly to the designated storage. This enhances security and ensures controlled access to the document upload functionality.
- **lambda-textract** This function is designed to handle document processing and indexing within our system. Triggered by an S3 event, it initiates document text detection using Amazon Textract, breaking the content into manageable chunks. Meta's RAG Tokenizer and Dense Passage Retrieval based text encoder are hosted in an EKS cluster together. This converts the text chunks into embeddings, representing the document's content numerically. The Lambda function then generates a unique document hash using SHA-256 for integrity verification. The embeddings and corresponding text and document information are indexed into an Amazon OpenSearch cluster. Simultaneously, the Lambda function stores metadata, including the document ID, user ID, and document URL, in a DynamoDB table for future reference.
- **authenticateUser** This Lambda function is responsible for handling user authentication using Amazon Cognito. When triggered by an API request containing a user's username and password, the function initiates an authentication request to the specified Cognito User Pool using the admin_initiate_auth method. If the authentication is successful, the function retrieves the JWT token from the response and decodes it using the JSON Web Key Set (JWKS) obtained from the Cognito User Pool's well-known endpoint. The decoded token contains user details, including the user's unique identifier. The function then constructs a response containing the user ID and returns a status code of 200. In case of authentication failure, it handles the error and returns a response with a status code 401. Additionally, the function includes appropriate headers for Cross-Origin Resource Sharing (CORS) to facilitate client-side requests.
- **postToDynamo** This is a key component in our Question and Answer (Q&A) processing pipeline. Triggered by an incoming API request, the function first queries DynamoDB to retrieve existing metadata associated with a user's Q&A session, including the user ID, document ID, and previous Q&A entries. Then, it utilizes OpenSearch to perform a kNN search, extracting contextual information related to the user's document. This information and the user's new question are then passed to the language model through a dedicated API endpoint. The response from the language model is parsed, and the extracted answer is appended to the user's Q&A history. Finally, the updated Q&A data is stored back in DynamoDB.
- **Lambda_presign** This Lambda function is pivotal in facilitating secure document uploads within the system. Triggered by an API request, it extracts the specified document name from the path parameters. It then utilizes the S3 client to generate a pre-signed URL for the corresponding S3 bucket and document key, ensuring secure and time-limited access for uploading. The generated URL, encapsulated in the response body, is then returned to the user application.
- **fetchFromDynamo** This Lambda function is a component in the system's API layer, handling requests to retrieve user-specific Q&A data from DynamoDB. Triggered by an incoming API request containing the user ID as a query string parameter, it queries the specified DynamoDB table using the provided user ID. The query results, representing the user's Q&A entries, are then processed and returned as a JSON-formatted response. If the requested user ID is not found in the DynamoDB table, the function returns a 404 status code and an error message.

C. Data Layer

Our application's data layer employs DynamoDB and OpenSearch. We opted for DynamoDB, instead of a traditional SQL database due to its efficiency in read-and-write operations, and for retrieving and displaying question-and-answer histories associated with each document for every user. Additionally, DynamoDB's flexible schema provides an effective solution for our RAG-based system's database needs.

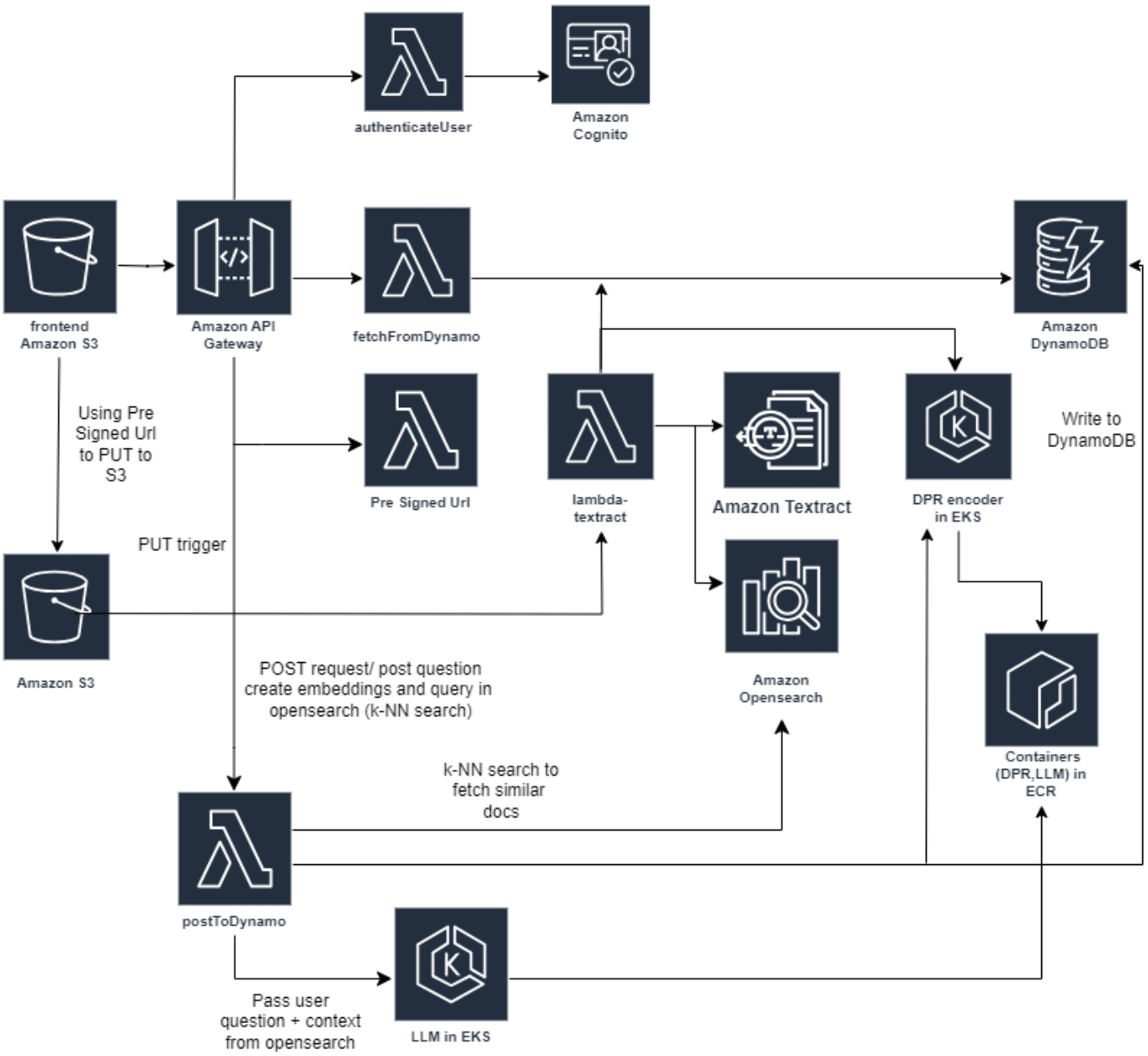


Fig. 1: Architecture Diagram

TABLE I: Schema of the DynamoDB Table: user-qna

Attribute	Type	Description
userId	string	Unique user ID
documentId	string	Document ID
documentUrl	string	S3 Document URL
qna	array	Array of question-answer pairs
qna.q		Question
qna.a		Answer

We chose OpenSearch for our RAG system due to its capability to execute cosine similarity-based k-nearest neighbor searches. This feature is essential for identifying text segments most relevant to a specific question within a document. OpenSearch's scalability and its reverse-indexing ability further enhance our system. These characteristics not only facilitate the analysis of vector quality and text chunk content but also support ongoing improvements and analytical assessments.

TABLE II: Data Schema for Index

Field	Data Type
embedding	knn_vector
chunk	string()
doc_id	string()

D. Logical Layer

The logical layer of our system is integral to the success of our information retrieval and generation framework. Large Language Models (LLMs), specifically NeuralHermes-2.5-Mistral-7B, forms the core of our approach. However, the inherent challenge with LLMs lies in their susceptibility to hallucination-generating inaccurate information due to a lack of awareness of the most recent data.

To combat hallucination, we adopted a Retrieval-augmented generation (RAG) framework, which involves providing the LLM with context through proper information retrieval. This involves using Facebook's query vectorizer and encoder to convert text content into searchable vectors for optimized indexing in OpenSearch.

For our LLM, NeuralHermes-2.5-Mistral-7B stands out due to its recent training on Intel's dataset, featuring 7 billion parameters. Its popularity stems from impressive performance and an active community fostering open collaboration. Notably, this model can be efficiently compressed to around 3GB through quantization while maintaining readability and compatibility with frameworks like PyTorch.

The logical layer's architecture incorporates the .GGUF format, allowing for readability and backward compatibility with various packages. Despite the quantization, prompt engineering is crucial for effective utilization since the model's efficiency is not equivalent to its full-form counterpart. However, the quantized model offers speed advantages and can even offload certain layers to a GPU if available.

To operationalize this logic, the model is hosted on AWS, taking into account constraints such as the AWS free tier. An EKS cluster is created, utilizing instances like m5.2X large for deployment. The architecture supports scalability by allowing multiple replicas or GPU instances. The Docker files and code for prompt engineering are well-defined, ensuring seamless execution of queries, providing context-aware answers or signaling when a question is outside the specified context.

In conclusion, the logical layer of our system is meticulously designed to address the challenges posed by LLMs, ensuring accurate, contextually relevant information retrieval and generation. The architecture, model selection, and deployment strategy on AWS collectively contribute to the efficiency and scalability of our information processing framework.

V. Future Work

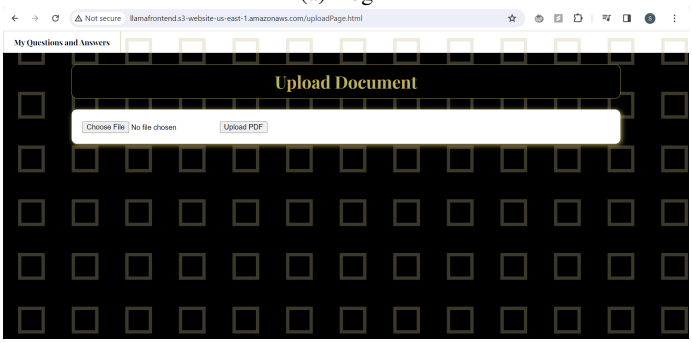
There are always opportunities for improvement and additional features to a project. Here are some suggestions for enhancing this application:

- **User Feedback and Rating System** Implement a system that allows users to provide feedback on the quality and relevance of the answers they receive. This data can be used to continuously improve the system and identify areas that may need enhancement.
- **Advanced User Authentication** Implement multi-factor authentication for enhanced security. This can include options such as SMS verification or biometric authentication, ensuring a more secure login process.
- **Personalized User Profiles** Create user profiles that store preferences. This can help tailor the user experience by providing personalized recommendations and insights based on their previous interactions.
- **Collaborative Questioning** Allow users to collaborate on questions by sharing them with colleagues or teammates. This feature can facilitate knowledge-sharing within organizations and improve collective decision-making.
- **Integration with External APIs** Integrate with external data sources or APIs to enrich the information retrieval process. This can include gathering real-time data, news updates, or information from industry-specific databases to enhance the system's knowledge base.
- **Data Visualization** Implement data visualization tools to present insights in a visually appealing manner. Graphs, charts, and other visual representations can help users better understand complex information.
- **Multi-language Support** Provide support for multiple languages to cater to a diverse user base. This can involve translating the user interface and ensuring that the system can understand and process queries in different languages.
- **Switch between different LLMs** Policies to allow multiple different LLMs to handle requests depending on the prompt/availability of the models.
- **LLM Output Streaming** Stream LLM Output to frontend instead of displaying after it finishes generating the response.

VI. Results

Our RAG AWS application has demonstrated notable efficiency in data retrieval and processing, leveraging the strengths of DynamoDB and OpenSearch. The significant improvement in response times and accuracy highlights the potential of combining NoSQL databases with advanced search technologies. As we continue to refine our approach, the focus will be on enhancing algorithms and user experience, cementing our position as a solution for complex data challenges.

(a) Login



(b) Upload Documents

Fig. 2: Ask Questions and Answers

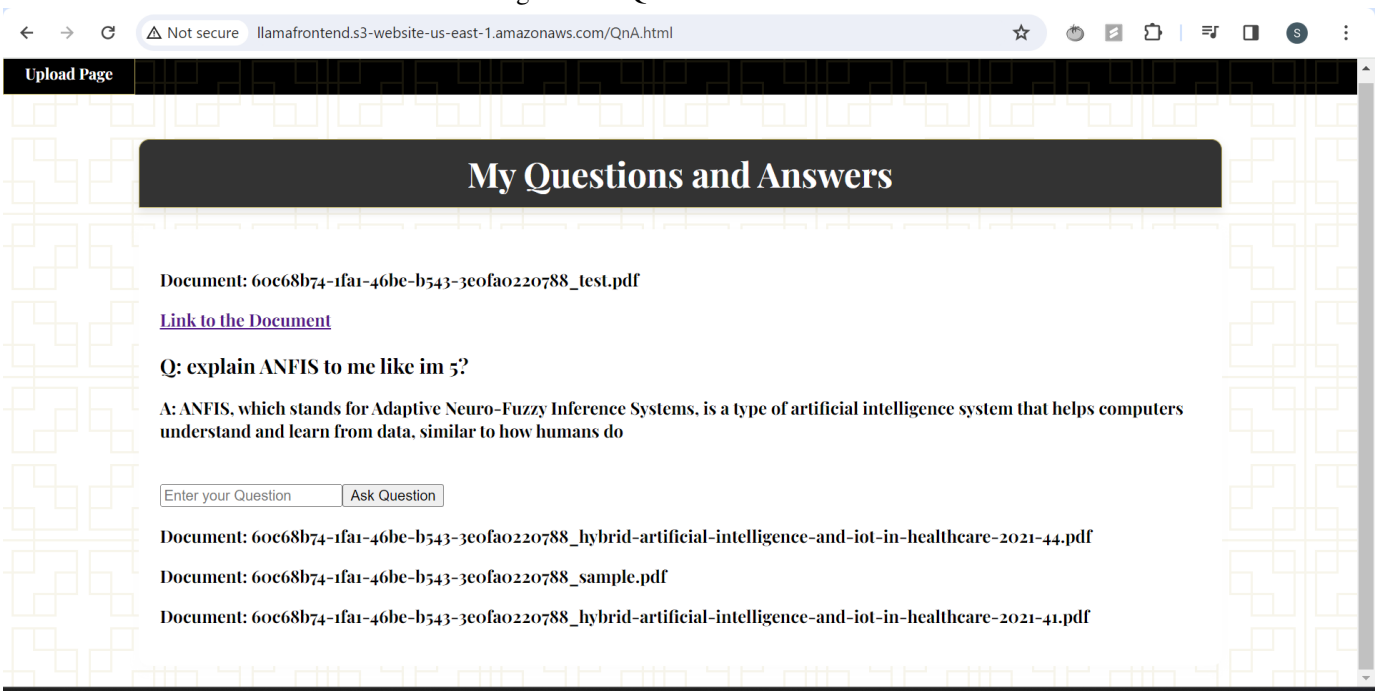


Fig. 3: Caption for Image 3

REFERENCES

[1] S. Riedel, "Retrieval Augmented Generation: Streamlining the creation of intelligent natural language processing models". Available: https://ai.meta.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/

[2] M. Chang, "REALM: Integrating Retrieval into Language Representation Models". Available: https://blog.research.google/2020/08/realm-integrating-retrieval-into.html

[3] Kensho Nerd. Available: https://nerd.kensho.com/

[4] Khan Academy. Available: https://blog.khanacademy.org/teacher-khanmigo/

[5] A. Piktus, Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks Available: https://arxiv.org/pdf/2005.11401.pdf