Indian Statistical Institute, Kolkata

Fund Fox India, A financial (Mutual Fund) chatbot

A dissertation submitted in fulfillment of the requirements for the award of
Master of Technology
in
Cryptology and Security

By:Sk Samiul Islam
CrS2215

Primary Supervisor:
Bhabesh Mazumdar
Franklin Templeton

Secondary Supervisor:
Dr. Debrup Chakraborty
Indian Statistical Institute,Kolkata

CERTIFICATE

This is to certify that the dissertation entitled **"Fundfox India, A Financial (Mutual Fund) chatbot"** submitted by **Sk Samiul Islam** to Indian Statistical Institute, Kolkata, in fulfillment for the award of the degree of **Master of Technology in Cryptology and Security** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

.....

Dr. Debrup Chakraborty,
Indian Statistical Institute, Kolkata

Acknowledgements

This internship has proved to be a great opportunity for me to expand my learning and professional horizon. I consider myself very lucky and honoured to have so many wonderful people lead me through, in this project.

I would like to show my cordial gratitude to my advisors, Bhabesh mazumdar, Vice President/Head of AI Products ,Franklin Templeton Investments also Venkat Rama Rao Nagulapally, Senior Data Scientist and Dr. Debrup Chakraborty, Professor, ISI Kolkata, for their guidance and support and encouragement throughout my internship. I would also like to thank my teammates in Franklin Templeton services for their valuable suggestions and discussions.They have made my works a lot easier to me with their important suggestions. Finally, I am thankful to my family and friends for their guidance.

Sk Samiul Islam
Indian Statistical Institute
Kolkata - 700108

Contents

Chapter 1

Introduction

1.1 Overview

During my internship at Franklin Templeton Investments, I had the opportunity to work on a pivotal project focused on the development of an advanced financial chatbot for Indian mutual funds. This chatbot is designed to enhance user interaction and provide accurate, real-time information regarding mutual fund investments. The project revolves around utilizing advanced language models, specifically GPT-4 from OpenAI, to deliver natural and context-aware responses to user queries. The aim is to improve the accessibility and understanding of mutual fund information for users, thereby supporting better investment decisions.

The development phase involved setting up backend and frontend codes, integrating the Azure Python SDK for initial setup, and leveraging AI search capabilities in Azure to create a robust search service. This includes setting up data source, indexe, skillsets, and indexer. Furthermore, we explored incorporating Document Intelligence to convert PDFs to JSON, aiming to ensure better data handling and response accuracy.

This project report outlines the methodologies used in developing the financial chatbot, discusses the challenges faced during the process, and describes the solutions adopted to address these challenges. Additionally, it emphasizes the crucial role this chatbot plays in improving user interaction and providing accurate financial information, thus contributing significantly to the users' investment decisions and financial knowledge.

1.2 Scope of the Project

- Understanding the requirements and features necessary for the chatbot to effectively respond to user queries about Indian mutual funds.
- Conducting detailed data analysis on the JSON data from Document Intelligence to identify patterns, relationships, and key insights in the data.
- Utilizing the GPT-4 model from Azure, along with several other Azure services, such as Document Intelligence, AI Search, Blob Storage, and Search Service.
- Evaluating the chatbot's performance using key metrics such as response accuracy, relevance, and user satisfaction.
- Continuously monitoring and improving the chatbot's capabilities by incorporating user feedback and new technological advancements.

1.3 Key Questions

1. How can we accurately interpret and respond to user queries in the context of Indian mutual funds?
2. What are the most effective methods to preprocess and transform financial data in JSON format got from document intelligence to make it suitable for AI search?
3. How do we ensure the chatbot's responses are accurate and reliable through appropriate evaluation metrics and validation techniques?
4. What strategies can be implemented to enhance the chatbot's user interface and user experience?

.

Chapter 2

Background

2.1 Financial Chatbots

Financial chatbots have emerged as a transformative technology in the financial services industry, providing enhanced customer service and operational efficiency. These chatbots use natural language processing (NLP) to interact with customers, answer queries, provide financial advice, and perform transactions. They are designed to understand and process user inputs, deliver relevant information, and facilitate various customer services through conversational interfaces.

2.2 Indian Mutual Funds

The mutual fund industry in India has experienced significant growth over the past few decades, driven by increasing investor awareness and favorable regulatory changes. Mutual funds pool money from multiple investors to invest in diversified portfolios of stocks, bonds, and other securities, providing retail investors with access to professionally managed portfolios.
The Indian mutual fund market is regulated by the Securities and Exchange Board of India (SEBI), ensuring transparency and protecting investor interests. With a wide range of fund categories, including equity, debt, hybrid, and sector-specific funds, mutual funds cater to varying investment goals and risk appetites.

2.3 Natural Language Processing

Natural Language Processing (NLP) is a branch of artificial intelligence that focuses on the interaction between computers and human language. NLP techniques enable machines to understand, interpret, and generate human language, facilitating seamless communication between humans and computers.

2.3.1 GPT-4 and its Applications

GPT-4 (Generative Pre-trained Transformer 4) is one of the most advanced language models developed by OpenAI. It is designed to understand and generate human-like text based on the input it receives. GPT-4 has numerous applications, including text summarization, translation, question answering, and conversational agents like chatbots.

In the context of financial chatbots, GPT-4 enhances the ability to process complex financial queries, provide accurate responses, and improve user interaction.

2.4 Azure Services

Microsoft Azure offers a comprehensive suite of cloud services that support the development and deployment of AI-powered applications, including financial chatbots. Azure provides robust infrastructure, advanced AI models, and integration capabilities to enhance chatbot functionality.

2.4.1 Azure AI Search service

Azure AI Search enables the creation of powerful search experiences by indexing and querying large volumes of data. It supports advanced search features such as natural language search, faceted navigation, and relevance tuning, making it ideal for developing intelligent financial chatbots that can quickly retrieve relevant information from vast datasets.

2.4.2 Blob Storage Service

Azure Blob Storage is a service for storing large amounts of unstructured data, such as text or binary data. It is highly scalable and designed for applications that require large volumes of storage, such as document archiving, backup, and big data analytics. In the context of financial chatbots, Blob Storage can be used to store large datasets, documents, and other resources needed to provide accurate and relevant responses to user queries.

2.4.3 Azure OpenAI Service

Azure OpenAI Service provides access to OpenAI's powerful models, including GPT-4, through the Azure platform. This integration allows developers to leverage advanced language models to enhance their applications with natural language understanding and generation capabilities. For financial chatbots, Azure OpenAI can be used to process and generate responses to complex financial queries, ensuring accurate and contextually relevant information is delivered to users.

2.4.4 Document Intelligence Service

Azure Document Intelligence leverages AI to extract data from various document formats, such as PDFs, images, and scanned documents. By converting these documents into structured JSON data, it facilitates efficient data processing and analysis, which is crucial for understanding and responding to user queries in a financial context.

Chapter 3

Project Methodology

3.1 Data Collection and Preparation

3.1.1 Data Sources

The data for this project was sourced from the official website of Franklin Templeton Investments India, specifically from the fund literature section here. This section includes various fund brochures and factsheets in PDF format. These documents provide comprehensive details about the different mutual funds offered by Franklin Templeton, including performance metrics, portfolio composition, investment strategies, and other essential information.
The PDFs contained a rich variety of data types, including:

- **Tables:** Detailing fund performance, fees, and expenses.
- **Paragraphs:** Descriptive text explaining fund strategies, market outlook, and other narratives.

3.1.2 Data Extraction Process

The process of extracting data from these PDFs involved several steps to ensure accuracy and completeness:

Download and Storage

- All relevant PDF documents were downloaded from the Franklin Templeton India website.
- These files were stored in Azure Blob Storage to facilitate easy access and management during the data extraction process.

Conversion to JSON (For experiments to improve result)

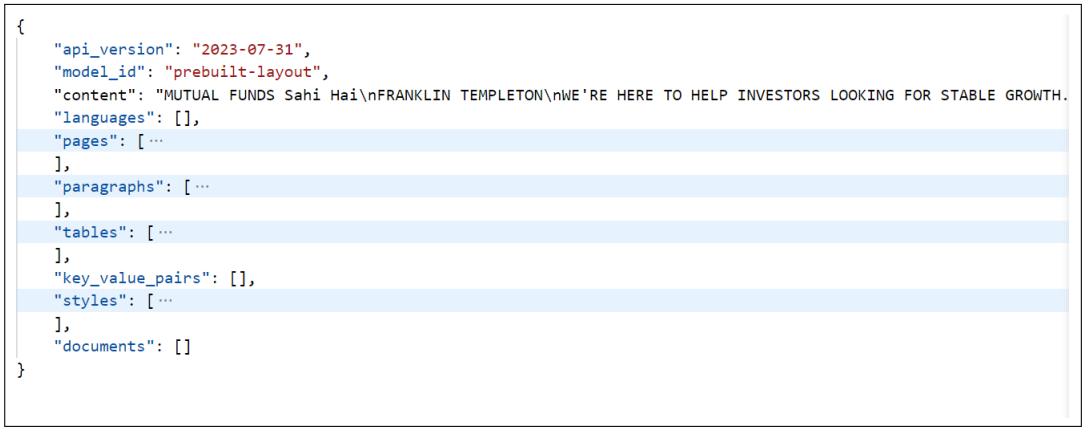
- Utilizing Azure Document Intelligence SDK, the PDF documents were converted into JSON format. This conversion was crucial as it transformed the unstructured data within the PDFs into a structured format, making it easier to process and analyze.

- Document Intelligence leveraged advanced AI techniques to accurately extract data from tables, diagrams, and text, ensuring that the context and relationships within the data were preserved.

Data Cleaning and Preprocessing (For JSON files)

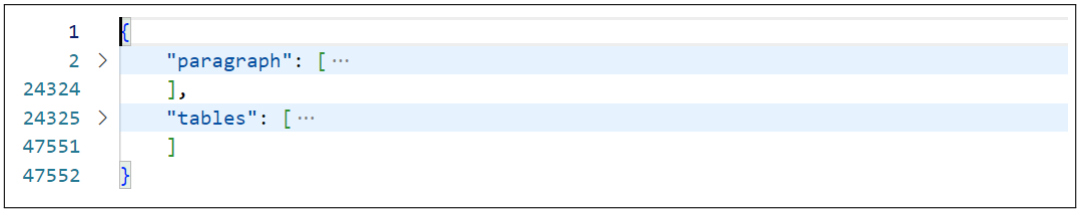
- To ensure the highest quality of data for our chatbot, we conducted several experiments to refine the structure of the JSON files and explored different methods of data extraction.

- In the **first approach**, we utilized the entire JSON file exactly as it was received from the document intelligence system. This allowed us to leverage the complete set of extracted information, maintaining the original format and structure provided by the extraction tool.



First Approch

- In the **second approach**, we focused on extracting only the paragraphs and tables from the JSON files. This involved creating a new JSON file that contained just these elements, thereby filtering out any other content that might not be directly relevant to our chatbot's requirements.

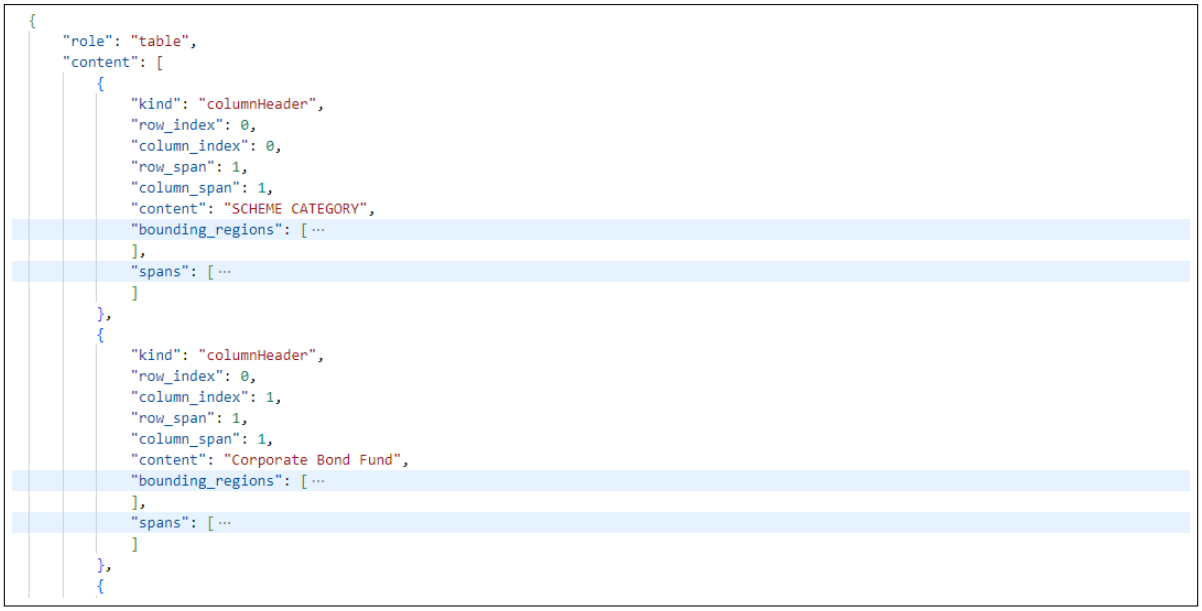


Second Approch

- In a **third method**, we created a list of dictionaries where each dictionary contains two items: 'role' and 'content'. In the first dictionary, 'role' is 'paragraph' and 'content' is the paragraph we obtained from document intelligence. In all the other dictionaries, 'role' is 'table' and 'content' is the 'cell' part from the tables we obtained from document intelligence.

Third Approch

'Paragraph' structure of 3rd approch



'Table' structure of 3rd approch

3.2 Integration with Azure AI Search

To enable efficient search capabilities within the chatbot, the data was to be integrated with Azure AI Search. This involves: Creating datasource, Index, skillsets and indexer.

3.2.1 Data Source in Azure AI Search

- **Data Source**

  A data source in Azure AI Search provides the connection information required to pull data into a search index. It serves as the origin from which data is imported into the search service for indexing. Data sources are used by indexers to retrieve and refresh the data in the search index, either on-demand or on a scheduled basis.

- **Components of a Data Source**

  - **Name**: Serves as a unique identifier, which must be lowercase, start with a letter or number, and be less than 128 characters.
  - **Description**: An optional text field to describe the data source.
  - **Type**: Specifies the kind of data source, such as Azure SQL Database, Azure Blob Storage, or Cosmos DB.
  - **Credentials**: Include the connection string or other necessary credentials to access the data source, with different formats required depending on the type.
  - **Container**: Defines the specific data to be indexed, such as a table, view, collection, or blob container, and may include an optional query to filter or transform the data.
  - **Encryption Key**: An optional component for encrypting data at rest using customer-managed keys, configured with Azure Key Vault.

- **Usage of Data Source**

  Data sources enable the indexing and refreshing of data in a search index. They can be created or updated using the Azure AI Search REST API with HTTP POST or PUT requests. Data sources are essential for keeping the search index up-to-date with the underlying data changes.

3.2.2 Index and Indexing in Azure AI Search

- **Search Index**

  A search index is a collection of searchable content defined by a schema. It resides in the Azure AI Search service, distinct from primary data stores, ensuring quick response times. Data import into the index occurs after defining the schema.

- **Index Schema**

  - The schema determines the structure of documents within the index. Documents are analogous to rows in a database table. Each document consists of fields, with each field having specific attributes like data type, searchable, filterable, sortable, etc.

- **Field Attributes**

  - **Searchable**: Enables full-text search on the field. Example: Splits "sunny day" into "sunny" and "day".
  - **Filterable**: Used in filter queries. Exact match required for strings.
  - **Sortable**: Allows sorting based on field values.
  - **Facetable**: Supports category-based hit count presentations.
  - **Key**: Unique identifier for documents within the index.
  - **Retrievable**: Determines if the field is returned in search results.

- **Index Management**

  - The physical structure and size of the index are managed by Azure. Index size is influenced by the number of documents, field attributes, and configurations like suggesters. An index is always available for querying, with real-time updates. Index isolation means each index operates independently. Endpoint connections and security are managed through specific URLs and roles for access control.

3.2.3 Skillsets

Skillset Concepts

- A skillset is a reusable object attached to an indexer in Azure AI Search, containing one or more skills that call built-in AI or external custom processing over documents from an external data source.
- Skills read from and write to an enriched document in memory, which starts as raw content and gains structure through skill execution.
- Outputs from skills are mapped to fields in a search index.

Skillset Definition

- A skillset is an array of skills that perform tasks like translating text or performing OCR on images.

- Skills have a context (scope of operation), inputs (from nodes in an enriched document), and outputs (new nodes in the document).

Skill Context

- The context determines how many times a skill executes and where in the enrichment tree the outputs are added.
- Context affects the shape of inputs; for example, if the context is a collection, the skill executes for each item in the collection.

Skill Dependencies

- Skills can execute independently or sequentially, where the output of one skill becomes the input of another.

Enrichment Tree

- An enriched document is a temporary, tree-like data structure created during skillset execution.
- It collects all changes introduced by skills and includes nodes for unenriched fields from the data source.
- Nodes in the enrichment tree can be selectively persisted to the index or knowledge store.

Indexer Definition

- An indexer uses properties and parameters to configure its execution, including mappings that set the data path to fields in a search index.
- There are two sets of mappings: **fieldMappings** for source fields to search fields, and **outputFieldMappings** for nodes in an enriched document to search fields.

Enrichment Example

- An example using hotel reviews demonstrates how an enrichment tree evolves through skill execution, including skills for text splitting, language detection, sentiment analysis, and key phrase extraction.

3.2.4 Indexer

An **indexer** in the context of search services, such as those provided by Azure Search, is a component that automates the process of indexing data from various data sources into a search index. The indexer reads data from a specified data source, transforms it as needed, and then loads it into the search index. This process can be scheduled to run periodically to ensure that the search index remains up-to-date with the latest data.

Key Features of an Indexer

- **Data Source Integration:** Indexers can connect to a variety of data sources such as Azure Blob Storage, Azure SQL Database, and Cosmos DB.

- **Data Transformation:** During the indexing process, indexers can transform data using predefined or custom scripts to match the schema and requirements of the search index.
- **Scheduling:** Indexers can be scheduled to run at specific intervals to keep the search index updated with new or modified data.
- **Error Handling:** Indexers provide mechanisms to handle errors and log details about indexing operations for troubleshooting.
- **Incremental Indexing:** Indexers can perform incremental indexing, which updates only the changed data rather than reindexing the entire data set, improving efficiency.
- **Field Mapping:** Indexers support field mapping, allowing you to map fields from the data source to the fields in the search index.
- **Data Deletion Detection:** Some indexers can detect deletions in the data source and update the search index accordingly to remove stale data.

Benefits of Using an Indexer

- **Automation:** Automates the process of keeping the search index synchronized with the data source.
- **Efficiency:** Reduces the manual effort required to index data, allowing for more efficient use of resources.
- **Consistency:** Ensures that the search index is consistently updated, providing accurate and up-to-date search results.
- **Scalability:** Can handle large volumes of data and scale according to the needs of the application.

Using an indexer simplifies the process of maintaining a search index, ensuring that it remains current with minimal manual intervention. This is particularly useful for applications that require robust search capabilities over large and dynamic data sets.
Through these steps, the project ensured that the financial chatbot could provide accurate, contextually relevant responses to user queries about Indian mutual funds, leveraging the comprehensive data available from Franklin Templeton's fund literature.

3.3 Conducting Several Experiments

3.3.1 Experiment 1: Indexing PDFs

In this experiment, we focused on indexing PDF documents directly from the blob storage. We created a data source and an indexer with specific skillsets such as chunking, overlapping, and vectorizing. Once the index was created, we conducted several queries on the fund brochures and factsheets using the OpenAI SDK in Azure. This integration allowed us to achieve highly accurate results with a commendably low response time. The process involved:

- **Data Source Creation:** We established a connection to the blob storage where the PDFs were stored.
- **Indexer Setup:** The indexer was configured to utilize chunking, overlapping, and vectorizing techniques to process the content of the PDFs efficiently.
- **Skillsets:**
  - *Chunking:* This helped in breaking down the documents into manageable pieces for easier processing.
  - *Overlapping:* This ensured that contextual information was preserved across chunks.
  - *Vectorizing:* This transformed the chunks into vectors, making them suitable for querying with the OpenAI SDK.
- **Query Execution:** We integrated the index with OpenAI and performed several queries related to fund brochures and factsheets.

The results from this experiment were quite satisfactory, with high accuracy and swift response times, demonstrating the effectiveness of our approach.

3.3.2 Experiment 2: Indexing Raw JSONs

In this experiment, we aimed to index JSON documents obtained from document intelligence directly from the blob storage. We created a data source and an indexer with skillsets focused on chunking and overlapping. After creating the index, we performed several queries on the fund brochures and factsheets using the OpenAI SDK in Azure. However, this time, the results were less accurate. The steps involved were:

- **Data Source Creation:** We connected to the blob storage where the JSON documents were stored.
- **Indexer Setup:** The indexer was configured with chunking and overlapping techniques.
- **Skillsets:**
  - *Chunking:* This helped in dividing the JSON documents into smaller, more manageable parts.
  - *Overlapping:* This ensured some contextual information was retained across different chunks.
- **Query Execution:** We integrated the index with OpenAI and performed several queries related to fund brochures and factsheets.

The majority of the queries returned incorrect answers, specially quaries frim tables, indicating that the direct use of raw JSON data should be structured in some suitable way for indexing.

3.3.3 Experiment 3: Indexing Processed JSONs

In this experiment, we processed the JSON documents obtained from document intelligence to extract only paragraphs and tables. We created a data source and an indexer with chunking and overlapping skillsets. After creating the index, we queried the fund brochures and factsheets using the OpenAI SDK in Azure. The steps were:

- **Data Source Creation:** We connected to the blob storage containing the processed JSON documents.
- **JSON Processing:** We extracted only paragraphs and tables from the raw JSON data.
- **Indexer Setup:** The indexer was configured with chunking and overlapping techniques.
- **Skillsets:**
  - *Chunking:* This helped in breaking down the documents into smaller parts for easier processing.
  - *Overlapping:* This retained some contextual information across different chunks.
- **Query Execution:** We integrated the index with OpenAI and performed several queries on the fund brochures and factsheets.

Again, the results were not that accurate, with most queries returning incorrect answers, indicating that further refinement was necessary.

3.3.4 Experiment 4: Indexing Structured JSONs

In this experiment, we structured the JSON data into a list of dictionaries where each dictionary contained two items: 'role' and 'content'. The first dictionary's 'role' was 'paragraph' with its 'content' being the paragraph extracted from document intelligence. The subsequent dictionaries had 'role' set to 'table' and 'content' set to the cell parts from the tables extracted from document intelligence. We then created a data source and an indexer with chunking and overlapping skillsets. After creating the index, we queried the fund brochures and factsheets using the OpenAI SDK in Azure. The steps involved were:

- **Data Source Creation:** We connected to the blob storage containing the structured JSON documents.
- **JSON Structuring:** We created a list of dictionaries, categorizing content into paragraphs and tables.
- **Indexer Setup:** The indexer was configured with chunking and overlapping techniques.
- **Skillsets:**
  - *Chunking:* This broke down the structured data into smaller parts for easier processing.

  - *Overlapping:* This retained some contextual information across different chunks.

- **Query Execution:** We integrated the index with OpenAI and performed several queries on the fund brochures and factsheets.

The results were still not upto the mark, with most queries returning incorrect answers. This suggests that the structuring approach and the existing skillsets might require further optimization or a different strategy altogether to improve accuracy and reliability.
In conclusion, while our **initial experiment with PDFs yielded positive results**, the subsequent experiments with JSON data highlighted challenges in achieving the same level of accuracy. Further investigation and optimization are necessary to refine our approach and enhance the performance of our indexing and querying processes using the OpenAI SDK in Azure.

3.4 Parsing

Parsing involves analyzing a string or text to extract meaningful information and convert it into a structured format. In the context of indexing JSON blobs in Azure AI Search, parsing modes are used to define how JSON documents are processed. Modes like `json`, `jsonArray`, and `jsonLines` determine whether the JSON blob is treated as a single document, an array of documents, or multiple entities separated by newlines. This ensures each JSON structure is optimally indexed for search and retrieval.
However, in our case, we did not employ parsing because our JSON data did not conform to the expected formats listed below:
JSON documents (one per blob):

```
{
    "article" : {
        "text" : "A hopefully useful article explaining how to pars
        "datePublished" : "2020-04-13",
        "tags" : [ "search", "storage", "howto" ]
    }
}
```

JSON arrays:

```
[
    { "id" : "1", "text" : "example 1" },
    { "id" : "2", "text" : "example 2" },
    { "id" : "3", "text" : "example 3" }
]
```

Nested JSON arrays:

```
{
    "level1" : {
        "level2" : [
            { "id" : "1", "text" : "Use the documentRoot property"
            { "id" : "2", "text" : "to pluck the array you want to
            { "id" : "3", "text" : "even if it's nested inside the
        ]
    }
}
```

The JSON structures in our data are highly complex and do not match these formats. Therefore, instead of utilizing parsing via the skillset, we treated them as markdown language and indexed them using chunking and overlapping strategies.

Chapter 4

App Building

4.1 Django Overview

- **High-Level Python Framework**: Django is a powerful web framework that allows developers to build web applications rapidly and with a clean, pragmatic design. Its comprehensive set of features and utilities enable efficient development and deployment.
- **Model-View-Controller (MVC) Architecture**: Django follows the MVC pattern, which separates the code into models (handling the data and database interactions), views (managing the user interface), and controllers (managing the logic and user input processing). This separation promotes organized and maintainable code.

4.2 Creating a Chatbot Framework with Django

1. **Set Up Django Project and App**:
   - Begin by installing Django and creating a new project to serve as the foundation for your chatbot application.
   - Within this project, create a new app dedicated to handling the chatbot functionalities. This modular approach helps in organizing the codebase efficiently.
2. **Define Models**:
   - Use Django's Object-Relational Mapping (ORM) to define the models. These models will represent the structure of your data, including user inputs and chatbot responses, and will handle interactions with the database seamlessly.
3. **Create Views**:
   - Develop views to handle HTTP requests and responses. These views will process user inputs, invoke the chatbot logic, and generate appropriate responses to be sent back to the user.
4. **Implement Forms or APIs**:

- Use Django forms to manage user interactions on the web interface, or implement REST APIs using Django REST Framework to enable interaction via other interfaces like mobile apps or other web services.

5. **Create Templates**:

   - Leverage Django's template system to design and render the chatbot's user interface. Templates allow you to dynamically generate HTML pages that display user inputs and chatbot responses in an engaging format.

6. **Manage Sessions and User Authentication**:

   - Utilize Django's built-in middleware to manage user sessions and authentication. This ensures that the chatbot can maintain context across interactions and provide a secure and personalized user experience.

After Creating the chatbot named 'Fundfox' by using Django framework and initially hosting in my local machine it looks like this:

Chapter 5

Challenges

We face several challenges in our current setup. These challenges can be broadly categorized as follows:

5.1 Connecting OpenAI Studio to AI Search

When using OpenAI Studio to connect to AI Search, if private endpoints are enabled, we must be granted access by Microsoft. This is necessary for communication between the services, as OpenAI Studio resides in the Microsoft VNET and cannot communicate with AI Search without pre-approval.

5.2 Codework in Azure Databricks

All codework is performed in Azure Databricks. Also several permissions needed to access endpoint and keys of several azure services in databricks.

5.3 Mounting Blob Container

We had to mount the Blob container to the specific cluster we were using in Databricks. This process required various permissions to access endpoints and keys of several Azure services.

5.4 Framework Issues

It was almost infeasible to create a Django framework within Databricks. Consequently, the framework part is done in VS Code. The OpenAI SDK is utilized in the 'views.py' file of the Django framework, but there were endpoint and key accessibility issues encountered there.

5.5 Processing JSON Formatted Files

Processing JSON formatted files presented another significant challenge, particularly in extracting table data accurately. To address this, we conducted several experiments.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

The development of an advanced financial chatbot for Indian mutual funds during my internship at Franklin Templeton Investments aimed to enhance user interaction and provide real-time investment information. The project leveraged OpenAI's GPT-4 model to deliver natural, context-aware responses, thereby improving the accessibility and understanding of mutual fund details for users, supporting better investment decisions.
The project involved setting up both backend and frontend systems, integrating the Azure Python SDK for the initial setup, and utilizing AI search capabilities within Azure to create a robust search service. This included configuring data sources, indexes, skillsets, and indexers, and incorporating Document Intelligence to convert PDFs to JSON for better data handling and response accuracy.
Several experiments were conducted to refine the chatbot's data processing and indexing capabilities. Initially, PDF documents were indexed directly from Azure Blob Storage, yielding accurate and timely results. Subsequent experiments focused on indexing JSON data extracted from Document Intelligence. Different methods of structuring and processing the JSON data were tested, including extracting only relevant paragraphs and tables, and structuring JSON data into categorized dictionaries. However, these experiments faced challenges in maintaining the same level of accuracy as the initial PDF indexing.
The project also explored parsing methods to optimize JSON document processing, but due to the complex nature of the JSON structures, parsing was not employed. Instead, chunking and overlapping strategies were used to index the data, treating it as markdown language.
The development process was supported by various Azure services, including Azure AI Search, Blob Storage, and Azure OpenAI Service, which facilitated the integration and deployment of the chatbot. The project demonstrated the potential of combining advanced AI models with robust cloud services to create intelligent, user-friendly financial tools, highlighting the need for ongoing optimization and refinement to achieve the desired accuracy and reliability in chatbot responses.

6.2 Future Plans for the Financial Chatbot Project

6.2.1 Improve Accuracy and Efficiency

- **Processing JSON Files Properly**: Enhance the accuracy of responses by refining the processing of JSON files.
- **Data Extraction**: Extract data from the top 5 documents to ensure comprehensive and relevant answers.
- **Utilize LangChain**: Integrate LangChain to handle complex questions and provide more accurate answers.

6.2.2 Integration of New Tools

- **Voice Recognition**: Incorporate GPT-4o for voice recognition to improve user interaction and accessibility.
- **Real-Time Market Data**: Integrate real-time market data to provide users with up-to-date information and trends.

6.2.3 Personalization Features

- **User Profiles**: Develop personalized user profiles to store preferences, investment goals, and past interactions. This will enable the chatbot to offer tailored responses and recommendations.
- **Customized Recommendations**: Use data from user profiles to provide customized mutual fund recommendations based on individual risk appetite, investment horizon, and financial goals.

6.2.4 Multilingual Support

- **Indian Languages**: Add multilingual capabilities to cater to users speaking different Indian languages, expanding the chatbot's accessibility and user base.

6.2.5 Predictive Analytics

- **Market Forecasts**: Incorporate predictive analytics to provide users with forecasts and trends based on current market data and historical patterns. This will help users make informed investment decisions.

6.2.6 Using Open AI to process JSON data

To effectively process table data from the JSON files, I will employ a strategy where each JSON-formatted table will be input into the OpenAI SDK. Using the advanced capabilities of OpenAI's language model, I will transform the tabular data into coherent paragraphs. This approach will ensure that the information is presented in a more natural and accessible format. Subsequently, these newly formatted files will be indexed and integrated into the system, enabling more efficient and accurate

data retrieval for the chatbot. This method will not only enhance the readability of the data but also improve the overall user experience by providing clear and concise information

6.2.7 Ongoing Development and Improvements

- **Query Handling**: Enhance query handling capabilities to cover a broader range of questions and improve response generation.
- **User-Friendly Interface**: Aim to create a highly accurate and user-friendly chatbot tailored to the Indian mutual fund market.

By implementing these future plans, the project aims to create a robust, efficient, and user-friendly financial chatbot that provides accurate and personalized assistance to users in the Indian mutual fund market.

References

[1] https://learn.microsoft.com/en-us/azure/ai-services/

[2] https://www.w3schools.com/django/index.php

[3] https://code.visualstudio.com/docs/python/tutorial-django