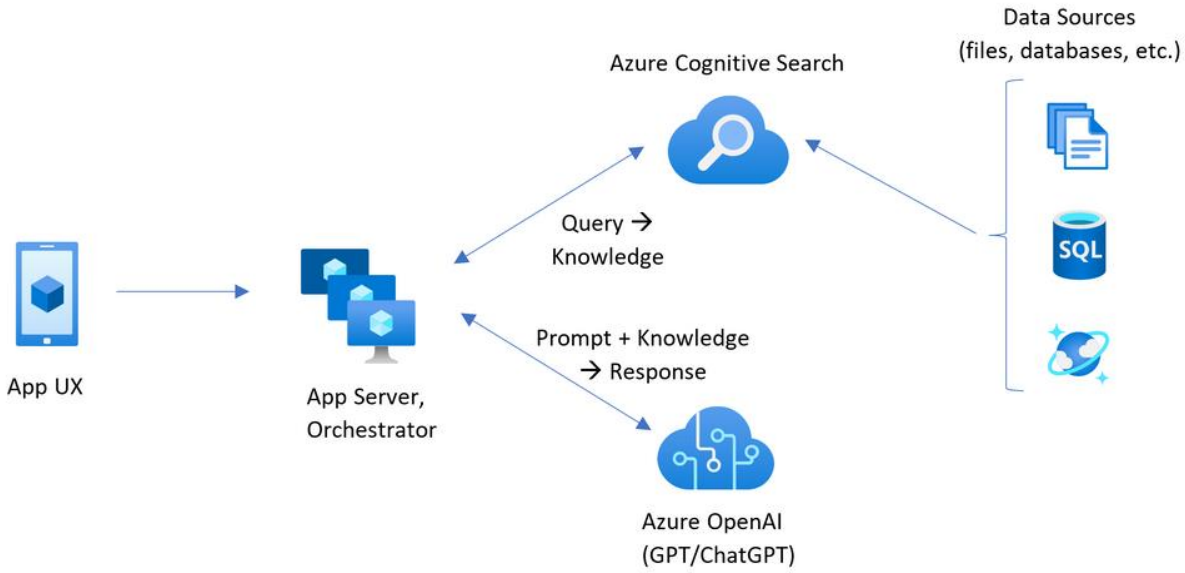
I have developed a system for creating ChatGPT-like experiences over our own data using the Retrieval Augmented Generation pattern. My solution seamlessly integrates the Azure OpenAI Service for ChatGPT and Azure Cognitive Search for data indexing and retrieval, enabling robust question-answering capabilities for financial data, with a specific focus on Honeywell's financial data.

The "Retrieval Augmented Generation" pattern combines the strengths of language models, such as ChatGPT, with efficient information retrieval from a specified dataset. This pattern ensures that responses are both linguistically coherent and grounded in real financial data, instilling trustworthiness in financial inquiries.

Here's a breakdown of my system's key components and functionalities:



1. Retrieval Augmented Generation Pattern: This approach combines ChatGPT's language capabilities with Azure Cognitive Search for document indexing and retrieval. It involves the following steps:

- User Query: Users pose questions about Honeywell's financial condition.
- Retrieval (Azure Cognitive Search): The system searches and retrieves relevant financial documents (e.g., annual reports) from the indexed dataset.
- Context Creation: The retrieved documents provide context for ChatGPT to understand the user's query.
- Response Generation: ChatGPT generates responses considering both the user's question and the retrieved documents' information.

2. Data Ingestion and Indexing: To make documents searchable, I've outlined the process of ingesting financial documents into Azure Cognitive Search:

- Data Format Conversion: Depending on the original data format, conversion into a suitable format like JSON or XML may be necessary.

- Index Schema Definition: A well-structured index schema, including fields like document title, content, and publication date, ensures efficient data retrieval.
- Data Indexing: Azure Cognitive Search processes the documents, tokenizes the text, and creates an optimized index for fast and relevant search queries.

3. User Interaction: Users can interact with the system through both chat and question-and-answer interfaces, offering flexibility in their interactions.

4. Trustworthiness Evaluation: The application provides tools for users to assess response trustworthiness. This is especially critical in the financial domain, where accuracy and reliability are paramount.

5. Data Preparation: Various data preparation approaches are demonstrated to ensure that financial data is structured and ready for querying. **Data Collection**: Obtained a dataset of financial documents from the U.S. Securities and Exchange Commission's EDGAR system. I am using a single document for now; annual report of Honeywell international inc. **Data Cleaning**: Clean the financial documents by removing special characters, converting text to lowercase, and addressing any other issues specific to your dataset. **Text Tokenization**: Tokenize the documents into sentences and words to facilitate text analysis.

6. Prompt Construction: I've provided guidelines or tools to assist users in constructing effective prompts for ChatGPT.

7. Model and Retriever Interaction: The system orchestrates the interaction between ChatGPT and Azure Cognitive Search, a crucial aspect of the retrieval-augmented generation pattern.

8. Customizable Settings: Users have the ability to customize the system's behavior to suit their specific needs, enhancing its utility.

9. Performance Tracing and Monitoring: Optional integration with Application Insights allows for performance tracing and monitoring, ensuring efficient system operation and insights into usage patterns.

By offering a sample dataset (Annual reports of Honeywell Inc from the SEC's EDGAR system) and a ready-to-use setup, I've made it easy for users to explore and leverage the capabilities of my system. In conclusion, my project is well-structured and provides a valuable tool for extracting insights from financial data, particularly for Honeywell, enhancing the user experience in this domain.

Here is the link to how you can try out the application that I created → https://app-backend-ceu547trycavw.azurewebsites.net/#/qa

Or, if you want to reproduce this code, follow these steps:

Local Environment Setup:

1. Install the required tools:
    - Azure Developer CLI
    - Python 3.9+
        - Ensure Python and pip are in the system's PATH (Windows users).
        - On Ubuntu, run **sudo apt install python-is-python3** if needed to link Python to Python3.
    - Node.js 14+
    - Git
    - Powershell 7+ (pwsh) - For Windows users only.
        - Make sure you can run **pwsh.exe** from a PowerShell terminal. Upgrade PowerShell if necessary.

Project Code Download:

1. Create a new folder and navigate to it in the terminal.
2. Run **azd auth login** to authenticate with Azure.
3. Run **azd init -t azure-search-openai-demo**. Note that this command initializes a git repository, so there's no need to clone an existing repository.

Deploying from Scratch: Execute the following command if you want to start from a fresh deployment:

4. Run **azd up**. This command provisions Azure resources and deploys the sample to those resources, including building the search index based on the files in the **./data** folder.
5. During this process, you'll be prompted to select two locations: one for the majority of resources and one for the OpenAI resource.

Accessing the Application:

6. After successful deployment, a URL will be printed to the console. Click on this URL to interact with the application in your web browser.

7. Please be patient, as it may take 5-10 minutes for the application to be fully deployed. If you encounter a "Python Developer" welcome screen or an error page, wait a bit and refresh the page.

Deploying with Existing Azure Resources: If you already have existing Azure resources, you can reuse them by setting environment values:

Existing Resource Group:

- Run **azd env set AZURE_RESOURCE_GROUP {Name of existing resource group}**
- Run **azd env set AZURE_LOCATION {Location of existing resource group}**

Existing OpenAI Resource:

- Run **azd env set AZURE_OPENAI_SERVICE {Name of existing OpenAI service}**
- Run **azd env set AZURE_OPENAI_RESOURCE_GROUP {Name of existing resource group where OpenAI service is provisioned}**
- Run **azd env set AZURE_OPENAI_CHATGPT_DEPLOYMENT {Name of existing ChatGPT deployment}** (Only needed if not using the default 'chat').
- Run **azd env set AZURE_OPENAI_EMB_DEPLOYMENT {Name of existing GPT embedding deployment}** (Only needed if not using the default 'embedding').

Existing Azure Cognitive Search Resource:

- Run **azd env set AZURE_SEARCH_SERVICE {Name of existing Azure Cognitive Search service}**
- Run **azd env set AZURE_SEARCH_SERVICE_RESOURCE_GROUP {Name of existing resource group with ACS service}**
- If the search service's location differs from the one you'll select for the **azd up** step, run **azd env set AZURE_SEARCH_SERVICE_LOCATION {Location of existing service}**.
- If the search service's SKU is not standard, run **azd env set AZURE_SEARCH_SERVICE_SKU {Name of SKU}** (See possible values).

Other Existing Azure Resources: You can also use existing Form Recognizer and Storage Accounts. Refer to **./infra/main.parameters.json** for a list of environment variables to pass to **azd env set** to configure those existing resources.

Provision Remaining Resources: 8. Now you can run **azd up**, following the steps in the "Deploying from Scratch" section above. This command will both provision resources and deploy the code.

Deploying Again:

- If you've only changed the backend/frontend code in the **app** folder, you don't need to re-provision Azure resources. Just run **azd deploy**.
- If you've changed the infrastructure files (**infra** folder or **azure.yaml**), you'll need to re-provision Azure resources by running **azd up**.