Retrieval Augmented Generation Industry Accelerator

Introduction

Retrieval Augmented Generation (RAG) is an AI framework for retrieving facts from an external knowledge base to ground large language models (LLMs) on the most accurate, up-to-date information and to give users insight into LLMs' generative process. It improves the quality of LLM-generated responses by grounding the model on external sources of knowledge to supplement the LLM's internal information. Implementing RAG in an LLM-based question answering system has two main benefits: it ensures that the model has access to the most current, reliable facts, and that users have visibility to the model's sources, ensuring that its claims can be checked for accuracy and ultimately trusted. In this accelerator we will:

- Connect to Elasticsearch and load some input data. This could be from a pdf or html pages as per user's requirement.
- Split and index the documents in Elasticsearch for search and retrieval.
- Deploy a python function which performs the RAG steps for a given input prompt and builds a response.
- Finally, a Question/Answer based interaction with the deployed function to accelerate the process end-to-end.

Tip: Download the PDF of these instructions from the Data assets section on the Assets page so you can keep these instructions open while you work.

Pre-Requisites

Ensure the following pre-requisites are in place to execute the accelerator:

1. An Elasticsearch instance with machine learning capability enabled. The Elasticsearch instance can either be on-premise, on Elastic Cloud, or on IBM Cloud (using the Databases for Elasticsearch service).
2. Within Elasticsearch, make sure either ELSER1 or ELSER2 spare encoder model is deployed.
3. The ability to execute prompts on Large Language Models such as llama-2-13b-chat or granite-13b-v2-chat using watsonx.ai.

Setup

Ensure the following setup is completed in your watsonx.ai platform:

1. In your project, setup connectivity to Elasticsearch:

    - If you are using a connection URL, username and password to connect to an Elasticsearch environment, create an Elasticsearch connection asset with these details in your project.
        - From the Project's **Assets** tab: **New Asset > Connection > Choose connection type of Elasticsearch**.
        - Enter elasticsearch_connect as the name of the connection.
        - Provide the connection URL, username, password, and SSL certificate.
        - Test the connection, then click Create to save to the project.
    - Alternatively, to connect to Elasticsearch using an API key, or connect to Elastic Cloud, specify the connection details in the parameter set.
        - From the Project's **Assets** tab: **Configurations > RAG_parameter_set**.
        - To connect to Elasticsearch using a connection URL and API Key, specify values in these parameters: elastic_search_url and elastic_search_api_key.
        - To connect to Elastic Cloud using a Cloud ID and API Key, specify values in these parameters: elastic_search_cloud_id and elastic_search_api_key.
2. Create a watsonx.ai deployment space, and update the parameter set with the Space ID:
    - Navigate to **Deployments > New Deployment Space**.
    - Provide a name for the deployment space. Also, specify a storage service and a machine learning service.
    - Click **Create**, then **View New Space**.
    - Select the **Manage** tab and copy the **Space GUID** value.
    - Back in the project, from the Project's Assets tab: **Configurations > RAG_parameter_set**.
    - Update the value for parameter watsonx_ai_space_id with the copied Space GUID value.
3. Ensure the Watson Machine Learning service is associated with the project (only required on watsonx.ai aaS):
    - From the Project's **Manage** tab, select **Manage > Services and Integrations > Associate Service** and select the Watson Machine Learning service.
4. Populate the remaining watsonx parameter values.
    - From the Project's Assets tab: **Configurations > RAG_parameter_set** - update the following parameter values:
    - watsonx_ai_url - hover over the region displayed in the top right header bar to identify your region (e.g. us-south, eu-de, etc.) - the corresponding URL is https://.ml.cloud.ibm.com, for example: https://us-south.ml.cloud.ibm.com.
    - watsonx_ai_api_key - you can create an API key by going to the Keys section of the Cloud console (only required on watsonx.ai aaS).
    - watsonx_ai_project_id - this can be found from the Project's **Manage** tab: **General** & **Project ID**.
5. Other parameters in the RAG Parameter Set and RAG Advanced Parameter Set contain default values. You can adjust these if you if required, or just go with the defaults.

Parameter Sets

Instead of entering variable factors as part of the code design, parameters are used to represent processing variables.

RAG Parameter Set

1. **elastic_search_connection_asset**: The name of the Elasticsearch connection asset in the project.

2. elastic_search_url: The URL endpoint for accessing Elasticsearch.
3. elastic_search_api_key: An API key for authenticating and authorizing access to Elasticsearch.
4. elastic_search_cloud_id: An identifier for an Elasticsearch instance on Elastic Cloud.
5. elastic_search_embedding_index_name: The name of the Elasticsearch index where embeddings are stored. The accelerator will create this index if it does not already exist.
6. elastic_search_source_index_name: The name of the Elasticsearch source data index. The accelerator will create this index if it does not already exist.
7. watsonx_ai_url: The URL for accessing the IBM watsonx.ai service.
8. watsonx_ai_api_key: An API key for accessing IBM watsonx.ai services (only required on watsonx.ai aaS).
9. watsonx_ai_project_id: An identifier for a watsonx.ai project that will run the LLM inferencing.
10. watsonx_ai_rag_space_id: The Deployment Space identifier for the IBM Watson Machine Learning space where the Q&A RAG python function will be deployed.
11. ingestion_document_name: The name of the file containing data for ingestion.

RAG Advanced Parameter Set

1. wml_rag_deployment_id: An identifier for the RAG deployed function. This will be populated by the notebook when the RAG python function is deployed.
2. elastic_search_source_index_mappings: A json file in the project that defines the structure mappings for a source index in Elasticsearch.
3. ingestion_chunk_size: The size of data chunks to be used when ingesting data into the Elasticsearch index.
4. ingestion_chunk_overlap: The amount of overlap between consecutive data chunks.
5. rag_es_top_n_results: The number of top results to retrieve from the Elasticsearch query.
6. elastic_search_model_id: An identifier for a specific model or configuration used within Elasticsearch, for search optimization or data processing.
7. elastic_search_pipeline_name: The name of the processing pipeline in Elasticsearch, defining a sequence of processing steps for data.
8. hallucination_threshold_max_text_overlap: A threshold for the maximum allowable overlap between generated text and source text used in hallucination detection.
9. hallucination_threshold_concatenated_text_overlap: A threshold for the allowable text overlap in concatenated text used in hallucination detection.
10. rag_es_min_score: The minimum score for results from the Elasticsearch query, results below this threshold will not be passed to the LLM.
11. elastic_search_embedding_index_mappings: A JSON file in the project that specifies the mappings for the destination index in Elasticsearch.
12. elastic_search_pipeline_processors: A JSON file in the project that defines the processing steps in the Elasticsearch pipeline.
13. elastic_search_template_file: A JSON file in the project that contains a template of predefined settings/configurations for Elasticsearch.
14. llm_prompt_template_file: A JSON file in the project that contains a template for prompts used in the LLM.
15. LLM_prompt_instructions_text: A text file that contains instructions for prompting the LLM.

Instructions for execution

Once the steps in the Pre-Requisites and Setup sections are complete, follow these steps to execute the accelerator:

1. Navigate to the **Assets** tab and scroll to the **Notebooks** section.
2. Edit the **Process HTML or PDF content** by clicking the edit icon that looks like a vertical ellipsis next to the notebook name. Follow the instructions in the notebook documentation to run the notebook cell by cell.
3. Edit the **Ingest into Vector Database** by clicking the edit icon that looks like a vertical ellipsis next to the notebook name. Follow the instructions in the notebook documentation to run the notebook cell by cell.
4. Edit the **Create and Deploy Q&A Python Function** by clicking the edit icon that looks like a vertical ellipsis next to the notebook name. Follow the instructions in the notebook documentation to run the notebook cell by cell.
5. Edit the **Q&A with RAG** by clicking the edit icon that looks like a vertical ellipsis next to the notebook name. Follow the instructions in the notebook documentation to run the notebook cell by cell.

Alternatively, execute via the **QnA Watson Pipeline**. This will run a number of notebook jobs in sequence.

1. Go to the **Assets Tab >> Under the Flows >> Open the QnA pipeline >> Run pipeline >> Trial run**. (If you wish to deploy the RAG function set the Deploy WML function pipeline parameter to True).
2. Edit the **Q&A with RAG** by clicking the edit icon that looks like a vertical ellipsis next to the notebook name. Then run the notebook cell by cell.

Sample data assets

- **data_for_ingestion.json**: Sample input json file created in **Process HTML or PDF content** notebook. This data in this file is ingested into the Elasticsearch indexes.
- **rag_llm_prompt_template.json**: Prompt template for the LLM. This has details related to which LLM to use, what type of sampling should be done, threshold information etc.
- rag_prompt_template_Llama2_RAG_instructions.txt: Instructions template for the prompt for llama2-13b-chat LLM.
- **elastic_search_source_mappings_template.json**: Template for source index mappings to be used for indexing.
- **elastic_search_embedding_mappings_template.json**: Template for destination index mappings to be used for indexing.
- **es_811_elser2_pipeline_processors_template.json** and **es_810_elser1_pipeline_processors_template.json**: Template for the ingestion pipeline. Templates are supplied for both ELSER 1 and ELSER 2.
- **elastic_search_ELSER_template.json**: The query template for Elastic Learned Sparse EncodeR search.
- **elastic_search_ELSER_BM25_hybrid_template.json**: The query template for Hybrid Elasticsearch combined with traditional vector search.
- **ibm-docs-SSYOK8.zip**: A sample zip file comprising 1147 HTML pages related to the documentation of the watsonx.ai. This can be used as sample data to run the accelerator.

Notebooks

Follow the instructions in the notebooks to step through the execution.

Process HTML or PDF content:

Process HTML or PDF content notebook processes 3 different types of data and convert data into a JSON format for subsequent ingestion into the vector index. The 3 different data types processed are:

1. PDF documents.
2. HTML files contained within a ZIP archive.
3. HTML content accessed via an IBM Product Documentation URL.

Ingest into Vector Database:

This notebook takes the intermediary JSON file produced by the **Process HTML or PDF content** notebook step and processes it into Elasticsearch indexes. There are a number of important parameters in the RAG parameter_set that need to be setup before executing the notebook, see the Setup section above: The main steps in the notebook are:

1. Connect to Elasticsearch and make sure ELSER is deployed.
2. Create the Elasticsearch source and destination indexes.
3. Create an ingest pipeline for ELSER.
4. Split documents into multiple segments with optimal overlap.
5. Load document segments into Elasticsearch content index.
6. Run ingest pipeline to vector index document segements.

Create and Deploy Q&A Python Function:

This notebook creates the Q&A RAG python function, and deploys the function, along with supporting assets to a deployment space. The main steps in the notebook are:

1. Initialize the deployment space.
2. Promote supporting assets to the deployment space.
3. Create RAG Scoring Pipeline Function.
4. Deploy the Scoring Function to the deployment space.
5. Test the scoring function by passing a question that will provide a response.
6. Update the value for parameter wml_rag_deployment_id with the deployment id of the python function. This is done for the parameter sets in both the project and deployment space.

Q&A with RAG:

This notebook can be used to try out the deployed function by asking some questions, either by executing a notebook, or in a more interactive mode using an ipywidget. Alternatively, you can test the deployed function is working by:

- Navigating to **Deployments > select the space where the function has been deployed > Deployments tab > watsonx-deployment**.
- From here you can see the endpoint. Select the **Test** tab and copy / paste the following into the input box, then **Predict**.

```
{
     "input_data": [
{
     "fields": ["Text"],
     "values": [["How do I add a database connection to a watsonx.ai project?"]]
   }]
}
```

Scripts

- rag_helper_functions.py script: Python script that provide many helper functions that are used throughout the notebooks.

Terms and Conditions