RAG System Documentation

Chiara Piccolo, Gioele Modica, Giorgio Angelo Esposito

Chapter 1

Documentation

1.1 generation Namespace Reference

Functions

- get_token ()
- check_aggregation (input_string, start_date, end_date)
- build_graph_from_json (json_file_path)
- describe_networkx_graph (G)
- periodic_read_kb ()
- read_kb ()
- faiss_generation ()
- extract_json_from_llm_response (response)
- parse_report_to_dict (report_text)
- generate_string (textual_response)
- generate_report (report)
- generate_dashboard (values, intro_string, x_axis_name, y_axis_name)
- response_creation (query, kpi_response, kpi_name)
- steps (query, context, date)
- user_query ()

Variables

- tuple chain1
- tuple chain2
- tuple chain3
- tuple chain4
- tuple chain5
- tuple chain6

1.2 Function Documentation

1.2.1 build_graph_from_json()

Description: This function builds a directed graph from a JSON file. The graph consists of: - **Nodes:** Representing machines and KPIs. - **Edges:** Representing the relationships where machines monitor specific KPIs.

Parameters:

- `json_file_path` (*str*): Path to the JSON file containing the graph data.

Returns:

- `G` (*networkx.DiGraph*): A directed graph containing machine and KPI nodes along with their relationships.

Example Usage:

```
from generation import build_graph_from_json
G = build_graph_from_json("data/graph_data.json")
```

Details: This function builds a directed graph from a JSON file. The graph contains nodes for machines and KPIs, and edges representing the relationships where machines monitor specific KPIs.

1.2.2 check_aggregation()

Description: This function checks whether the input string contains the words "daily," "monthly," or "weekly" (case-insensitive) and returns the first match. If no match is found, it returns `None`. Additionally, if the `start_date` is equal to the `end_date`, it defaults to returning "daily." If no specific match or single-day period is identified, the function returns "overall."

Parameters:

- `input_string` (*str*): The input string to analyze.
- `start_date` (*str*): The start date of the period under consideration.
- `end_date` (*str*): The end date of the period under consideration.

Returns:

- *str* or `None`: The matched time aggregation word (`"daily"`, `"monthly"`, or `"weekly"`), "daily" if the period is a single day, or "overall" otherwise.

Details: The function first scans the `input_string` for any of the specified aggregation keywords. If none of these words are found, it checks the provided dates. When the start and end dates are the same, it assumes "daily" aggregation. For multi-day periods without a specific match in the string, the function defaults to "overall."

1.2.3 describe_networkx_graph()

Description: This function generates a description of all nodes in a NetworkX graph. Nodes are categorized into machines and KPIs, and detailed descriptions are provided based on their attributes and relationships.

Parameters:

- `G` (*networkx.Graph*): A NetworkX graph containing nodes and edges with attributes.

Returns:

- `descriptions` (*dict*): A dictionary where keys are node names, and values are their descriptions.

Details: The function processes each node in the graph, analyzing its type (machine or KPI) and extracting its attributes. Relationships between nodes (e.g., which machines monitor which KPIs) are included in the descriptions. This output can be used to understand the structure and relationships within the graph.

1.2.4 extract_json_from_llm_response()

Description: This function extracts relevant JSON data from an LLM response and formats it into a structured dictionary. Additionally, it retrieves the corresponding machine ID from the knowledge base (KB) based on the machine name.

Parameters:

- `response` (*str*): The raw response from the LLM, which contains JSON-like content.

Returns:

- `result` (*dict*): A dictionary containing extracted and formatted values for KPI queries, including the machine ID and default values for missing fields.

Details: The function processes the raw response by identifying and parsing the JSON content embedded in it. It ensures that any missing fields are assigned default values and attempts to map machine names to their respective IDs using a predefined knowledge base. The output dictionary is suitable for downstream tasks like KPI analysis or system monitoring.

1.2.5 faiss_generation()

Description: This function generates a FAISS vector store from a knowledge base (KB) JSON file. It follows a sequence of steps to process the data, convert it into embeddings, and build a FAISS index for efficient information retrieval.

Steps:

1. Build a graph from the KB JSON file.
2. Generate natural language descriptions for nodes in the graph.
3. Embed the descriptions using HuggingFace's Sentence-Transformers model.
4. Create a FAISS vector store with these embeddings for retrieval.
5. Save the FAISS vector store locally.

Returns:

- `embeddings` (*list*): A list of embeddings generated for node descriptions.
- `vector_store` (*FAISS*): The FAISS vector store object for efficient retrieval.

Details: The function starts by creating a graph from the provided KB JSON file. For each node in the graph, it generates a natural language description, which is then converted into embeddings using HuggingFace's Sentence-Transformers. These embeddings are stored in a FAISS index, which allows for fast similarity-based retrieval. Finally, the FAISS vector store is saved locally for future use in querying the knowledge base.

1.2.6 generate_dashboard()

Description: This function generates a structured dictionary for a dashboard visualization. The resulting dictionary contains data for axis labels, introductory text, and values that will be displayed in the dashboard.

Parameters:

- `values` (*list*): A list of values for the dashboard (e.g., data points).
- `intro_string` (*str*): Introductory text for the dashboard (e.g., "Here is the weekly production rate for Machine_X").
- `x_axis_name` (*str*): The label for the X-axis (e.g., "Date").
- `y_axis_name` (*str*): The label for the Y-axis (e.g., "KPI name" or "value type").

Returns:

- `dashboard` (*dict*): A structured dictionary containing the dashboard data, including axis labels, introductory text, and values to display.

Details: The function takes the provided values and text and organizes them into a dictionary format suitable for rendering a dashboard. This includes adding axis names, a description of the data, and the actual data points to be visualized. The function's output can be used for dynamic generation of dashboards or reports.

1.2.7 generate_report()

Description: This function generates a structured dictionary for a report. It processes the provided report content and organizes it in a standardized format, allowing for easy inclusion of optional elements such as additional text or dashboard components.

Parameters:

- `report` (*dict*): The report content to include, structured as a dictionary.

Returns:

- `structured_report` (*dict*): A structured dictionary containing the report data and placeholders for optional text or dashboard elements.

Details: The function processes the provided dictionary containing the report's core content and formats it into a structure suitable for various report generation needs. The output can also include placeholders that allow the integration of additional elements such as text or dashboards.

1.2.8 generate_string()

Description: This function generates a structured dictionary for a text response. The output dictionary includes the provided textual message and optional placeholders for additional components such as dashboards or reports.

Parameters:

- `textual_response` (*str*): The textual message or response to be included.

Returns:

- `structured_response` (*dict*): A structured dictionary containing the text response and placeholders for optional dashboard or report data.

Details: The function processes the provided textual message and organizes it into a dictionary format. This format allows easy integration with other elements, such as dashboards or reports, and ensures the response is structured for further use.

1.2.9 get_token()

Description: This function retrieves the token value from the API by sending a POST request with the username and password. The token is used for authentication in subsequent API requests.

Returns:

- `token` (*str*): The token value retrieved from the API.

Details: The function sends a POST request with the necessary credentials (username and password) and receives a token in response. This token is then returned for use in authenticating further requests to the API.

1.2.10 parse_report_to_dict()

Description: This function parses a text-based report and converts it into a structured dictionary with a title and sections. The dictionary organizes the report content for easier manipulation and access.

Parameters:

- `report_text` (*str*): The report content in text format to be parsed.

Returns:

- `report_dict` (*dict*): A dictionary containing the title and sections of the report.

Details: The function processes the raw report text, identifies key components (such as title and sections), and converts them into a structured dictionary format. This allows for better organization and easier extraction of relevant information from the report.

1.2.11 periodic_read_kb()

Description: This function runs in an infinite loop, periodically calling the `read_kb` function to fetch and update the knowledge base every 3600 seconds (1 hour). It is designed to be run as a daemon thread, ensuring non-blocking execution in the main program.

Details: The function is implemented to continuously fetch updates from the knowledge base at regular intervals. It operates in the background as a daemon, meaning it will not block the main program's execution. The periodic interval for fetching the knowledge base is set to one hour (3600 seconds), but it can be adjusted as needed.

Behavior:

- Periodically invokes the `read_kb` function.
- Executes in a non-blocking manner using a daemon thread.
- Designed to run indefinitely, ensuring continuous updates.

1.2.12 read_kb()

Description: This function fetches the knowledge base (KB) from a remote API and updates the local KB file if any changes are detected.

Details:

- It compares the API response with the existing local KB file.
- If the content differs, it updates the file and triggers further processing.

Returns:

- The result of the `faiss_generation` function if updates occur.
- `None` if no update is needed or an error occurs.

1.2.13 response_creation()

`generation.response_creation(query, kpi_response, kpi_name)`

Description: Processes the response from the KPI engine and generates an appropriate structured response based on the query and KPI values.

Arguments:

- `query` (`str`): The query provided by the user.
- `kpi_response` (`dict`): The response from the KPI engine containing KPI values and units.
- `kpi_name` (`str`): The name of the KPI being processed.

Returns:

- `dict`: A structured dictionary that represents a text response, a report, or a dashboard based on the KPI data and query context.

1.2.14 steps()

`generation.steps(query, context, date)`

Description: This function orchestrates the retrieval-augmented generation (RAG) pipeline. It determines whether historical data is needed, processes queries, fetches KPI data, and generates appropriate structured responses.

Arguments:

- `query` (`str`): The user query or question.
- `context` (`str`): The context provided for processing the query.
- `date` (`str`): The current date for reference.

Returns:

- `dict`: A structured dictionary representing the final response, which can be a text, report, or dashboard.

1.2.15 user_query()

`generation.user_query()`

Description: Endpoint to handle user queries.

Behavior:

- Processes a POST request with a JSON body containing a `'query'` field.
- Utilizes a FAISS vector store and retrieval pipeline to return the most relevant results processed by the LLM.

Returns:

- `JSON`: A structured response based on the query, or an error message.

1.3 Chain Documentation

1.3.1 chain1

Purpose: Determines if historical data is required for a given query.

Prompt Used: `enough_info_prompt` - This prompt checks if the user's query requires access to historical data. It evaluates the context and the query and returns `yes` if historical data is needed, otherwise `no`.

Input:

- `context` - The context containing KPI and machine information.
- `query` - The user's query.

Return: `'yes'` or `'no'` indicating if historical data is required.

Initial value:

```
00001 =  (
00002     {
00003         'context': RunnablePassthrough(),
00004         'query': RunnablePassthrough()
00005     }
00006     | enough_info_prompt
00007     | model
00008     | StrOutputParser()
00009 )
```

1.3.2 chain2

Purpose: Processes informational queries and generates direct, concise responses.

Prompt Used: `informational_query_prompt` - This prompt provides a direct, concise response to an informational query. It utilizes the context to answer user questions without requiring historical data.

Input:

- `context` - The context retrieved from the Knowledge Base.
- `query` - The user's query.

Return: A concise answer to the user's query.

Initial value:

```
00001 =  (
00002     {
00003         'context': RunnablePassthrough(),
00004         'query': RunnablePassthrough()
00005     }
00006     | informational_query_prompt
00007     | model
00008     | StrOutputParser()
00009 )
```

1.3.3 chain3

Purpose: Extracts key query details, such as KPI, machine, and time range, from user queries.

Prompt Used: `action_query_prompt` - This prompt extracts key information from user queries, including KPI name, machine name, time range, and the type of operation requested (e.g., `sum`, `avg`, `max`, or `min`).

Input:

- `context` - The context retrieved from the Knowledge Base.
- `query` - The user's query.
- `current_date` - The current date for resolving relative dates.

Return: A JSON-formatted structure with query details.

Initial value:

```
00001 =  (
00002     {
00003         'context': RunnablePassthrough(),
00004         'query': RunnablePassthrough(),
00005         'current_date':  RunnablePassthrough()
00006     }
00007     | action_query_prompt
00008     | model
00009     | StrOutputParser()
00010 )
```

1.3.4 chain4

Purpose: Generates a single-value text response for a user query. This response is used when the system needs to provide a single KPI value for a given machine and time frame.

Prompt Used: `single_value_prompt` - This prompt generates a concise, text-based response when a query requests a single value. It returns the KPI value along with its unit of measurement.

Input:

- `query` - The user's query.
- `value` - The single KPI value returned by the KPI Calculation Engine.
- `unit` - The unit of measurement associated with the KPI.

Return: A concise text-based response with the KPI value and unit.

Initial value:

```
00001 =  (
00002     {
00003         'query': RunnablePassthrough(),
00004         'value': RunnablePassthrough(),
00005         'unit': RunnablePassthrough(),
00006     }
00007     | single_value_prompt
00008     | model
00009     | StrOutputParser()
00010 )
```

1.3.5 chain5

Purpose: Generates a one-sentence description of the content, typically for dashboards or summaries. It uses the user's query to generate a brief description that can be displayed along with dashboards.

Prompt Used: `description_prompt` - This prompt generates a short, one-sentence description that summarizes the content of a dashboard or a similar visualization. It uses the user's query to generate a relevant description.

Input:

- `query` - The user's query.

Return: A one-sentence description that provides context for the dashboard.

Initial value:

```
00001 =  (
00002     {
00003         'query': RunnablePassthrough()
00004     }
00005     | description_prompt
00006     | model
00007     | StrOutputParser()
00008 )
```

1.3.6 chain6

Purpose: Generates a comprehensive report in response to a user's query. The report includes a descriptive title, key performance metrics, an analysis of trends, and a review of anomalies. It provides a high-level overview of KPI performance for a specified machine and time frame.

Prompt Used: `report_prompt` - This prompt generates a full report based on user queries. It extracts the KPI data, analyzes key trends, and provides insights on the performance of machines over a specified period.

Input:

- `query` - The user's query requesting a report.
- `data` - The production data used to generate the report. This includes KPI values and timestamps.
- `unit` - The unit of the KPI being reported.

Return: A comprehensive report that includes a title, key metrics, trends, and observations. This report is formatted as structured text to facilitate display on exported documents.

Initial value:
```
00001 =  (
00002     {
00003         'query': RunnablePassthrough(),
00004         'data': RunnablePassthrough(),
00005         'unit': RunnablePassthrough(),
00006     }
00007     | report_prompt
00008     | model
00009     | StrOutputParser()
00010 )
```

1.4 Prompt Documentation

1.4.1 enough_info_prompt

Purpose: Determines whether historical data is needed to answer a specific query regarding Key Performance Indicators (KPIs).

Prompt Used: `enough_info_prompt` - Evaluates the query and context to decide if historical data is necessary for answering the question.

Input:

- `context` - The context containing existing KPI data.
- `query` - The question requiring evaluation.

Return: A logical evaluation ('yes' or 'no') with reasoning that clarifies if historical data is required.

Prompt details:

> You are an evaluator. Your job is not to answer the query. Your task is to determine if historical data, which consists only of previously recorded measurements of specific KPIs provided in the context, is needed to answer the query.
> **Important:**

- Historical data refers strictly to the past measurements of the KPIs explicitly provided in the context.
- Historical data does not contain information about whether a KPI exists if that KPI is not mentioned in the context. You cannot make assumptions or speculate about the existence of additional KPIs.
- If a KPI is explicitly mentioned in the context, you do not need historical data to confirm that it exists. Historical data is only useful for analyzing past measurements, trends, or comparing values over time for that KPI.

Historical data is needed only if the query requires:

- Analyzing trends or changes over time.
- Looking at historical records of measurements for KPIs that are explicitly mentioned in the context.

If the query is about confirming the existence of a KPI that is already provided in the context, historical data is not needed.
Explain your reasoning and conclude with a final answer: respond `yes` if historical data is needed, otherwise respond `no`. You must strictly adhere to this format.

Context: `{context}`. Question: `{query}`.

Answer:

1.4.2 informational_query_prompt

Purpose: Generates a direct and concise answer to a query using information provided in the context.

Prompt Used: `informational_query_prompt` - Responds to queries without introducing unnecessary information.

Input:

- `context` - The context containing relevant data.
- `query` - The user's question.

Return: A direct and concise answer to the query.

Prompt details:

Please answer the following question using the information in the provided context.
Your answer should be **direct and concise**, focusing specifically on addressing the question.
If the question asks for additional details, provide only the specific information requested. Do not introduce information or explanations beyond what is directly asked for in the question.

Context: `{context}`. Question: `{query}`.

Answer:

1.4.3 action_query_prompt

Purpose: Extracts specific details (timeframe, operations, KPIs, and machine names) from a query in a structured JSON format.

Prompt Used: `action_query_prompt` - Breaks down the query into components including timeframe, operation, KPI name, and machine name.

Input:

- `context` - Additional information relevant to the query.
- `query` - The user's query to analyze.
- `current_date` - Reference date for interpreting relative timeframes.

Return: A structured JSON response containing:

- `start_range` - Start of the timeframe.
- `end_range` - End of the timeframe.
- `operation` - Mathematical operation (e.g., sum, avg).
- `KPI_name` - Key Performance Indicator name.
- `machine_name` - Machine name.

Prompt details:

Analyze the provided query to extract specific details. Follow these steps precisely:

Step 1: Determine the Timeframe Identify the timeframe mentioned in the query. Classify it into one of the following categories:

- **Specific Date:** A single, specific point in time. Examples include:
  - "on September 15th"
  - "on 2023-09-15"
  - "yesterday"
  - "two days ago"
- **Range of Dates:** A continuous range that has a clear start and end date. Examples include:
  - "between September 1st and September 15th"
  - "over the past week"
  - "from July to August"

  Note: A range always involves a start and an end date, covering all the days in between.

Step 2: Identify the Operation Check if the query mentions any of the following operations:

- sum, avg, max, min.

If no operation is mentioned, leave this field as 'null'.

Step 3: Extract the Following Details: Extract and organize the information as described below:

- For a **Specific Date:** Provide it in the `start_range` field in `YYYY-MM-DD` format.
- For a **Date Range:** Provide both `start_range` and `end_range` fields in `YYYY-MM-DD` format.

- For an **Aggregation Period:** Provide `start_range` and `end_range` fields in `YYYY-MM-DD` format.
- `'operation'`: Specify the operation mentioned (e.g., `'sum'`, `'avg'`, `'max'`, `'min'`). If none, set this field to `'null'`.
- `'KPI_name'`: Identify the key performance indicator mentioned.
- `'machine_name'`: Determine the machine referred to in the query.

Return the extracted information in a structured format (`json`). Today's date is `{current_date}` for reference.

Query: `{query}`. Context: `{context}`.

Answer:

1.4.4 single_value_prompt

Purpose: Provides a concise answer to a query using a single provided value and its unit of measurement.

Prompt Used: `single_value_prompt` - Generates answers based on a specific value and its unit.

Input:

- `value` - The numerical value.
- `unit` - The unit of measurement for the value.
- `query` - The question requiring an answer.

Return: A concise answer addressing the query using the provided value and unit.

Prompt details:

Please answer the following question using the provided **value** and the given **unit of measurement**.

Your answer should be **direct and concise**, focusing specifically on addressing the question.
If the question asks for additional details, provide only the specific information requested.
Do not introduce information or explanations beyond what is directly asked for in the question.

Value: `{value}`.
Unit of measurement: `{unit}`.
Query: `{query}`.

Answer:

1.4.5 description_prompt

Purpose: Generates a concise one-sentence description based on the provided query.

Prompt Used: `description_prompt` - Converts the query into a brief descriptive sentence.

Input:

- `query` - The query describing the requested information.

Return: A single-sentence description summarizing the query.

Prompt details:

> Provide a **concise one-sentence description** of the content based on the given query.
>
> **Example 1: Query:** What was the working time for LaserCutter_1 over the past week? **Answer:** Below is the working time for LaserCutter_1 over the past week.
>
> **Example 2: Query:** Show a dashboard with the utilization rate for Machine_Z over the past week. **Answer:** Below is the dashboard with the utilization rate for Machine_Z over the past week.
>
> **Query:** `{query}`
> **Answer:**