AI VOID FIELD GUIDE

Step-by-Step Tutorials

Focused, practical tutorials that teach you how to accomplish specific tasks step by step. Learn by doing with clear instructions and working code examples.

AI VOID

2026-05-11 17:36 IST

Contents

01 Get Started with FalkorDB GraphRAG SDK 1.0 3

CHAPTER 01

Get Started with FalkorDB GraphRAG SDK 1.0

What you'll build: A basic GraphRAG application that leverages FalkorDB and an LLM to answer natural language queries from ingested data. **Time needed:** ~45 minutes **Prerequisites:** Python 3.10+, Running FalkorDB instance, LLM API Key (e.g., OpenAI, Anthropic) **Version used:** 1.0.0

Introduction to FalkorDB GraphRAG SDK 1.0

In the exciting world of Large Language Models (LLMs), one of the biggest challenges is ensuring they provide accurate, up-to-date, and contextually relevant information, rather than "hallucinating" or relying on outdated training data. This is where Retrieval Augmented Generation (RAG) comes into play. RAG empowers LLMs to retrieve information from an external knowledge base before generating a response, drastically improving accuracy and trustworthiness.

But what if your data is complex, interconnected, and full of rich relationships? Traditional RAG, often built on vector databases, can sometimes struggle to capture these nuances, leading to less precise answers. This is the problem GraphRAG solves.

The **FalkorDB GraphRAG SDK 1.0** is a powerful new tool designed to build intelligent GraphRAG applications with ease. It leverages the strengths of graph databases, specifically FalkorDB, to create a highly accurate and efficient retrieval layer for your LLM applications.

Why GraphRAG matters: GraphRAG excels by converting your unstructured text into a structured knowledge graph, where entities (like people, places, concepts) and their relationships are explicitly defined. When a query comes in, the SDK intelligently navigates this graph to find the most relevant, interconnected pieces of information, providing a far richer context to the LLM. This approach directly tackles common LLM issues like high token costs, slow responses, and inaccurate answers by ensuring the LLM receives precisely the right context. FalkorDB's SDK has even been recognized as a top performer on GraphRAG-Bench, highlighting its superior retrieval capabilities.

What problem it solves:

- **Hallucinations:** By grounding responses in your specific, accurate data.
- **Stale Information:** By always querying your live knowledge base.

- **Lack of Context:** By providing rich, interconnected information from the graph.
- **High Token Costs:** By retrieving only the most relevant information, reducing the amount of data sent to the LLM.
- **Slow Responses:** By optimizing the retrieval process through graph traversal.

This tutorial will guide you through setting up your environment, installing the SDK, understanding its core mechanics, and building your first GraphRAG application.

Prerequisites and Environment Setup

Before we dive into the code, let's ensure your development environment is ready. We'll need Python, a running FalkorDB instance, and your LLM API key.

1. **Python 3.10+:** Ensure you have Python 3.10 or newer installed. You can check your version by running:

   ```
   python3 --version
   ```

   If you need to install or update Python, refer to the official Python documentation.

2. **Create a Virtual Environment:** It's always a good practice to work within a Python virtual environment to manage dependencies for each project.

   First, create the environment:

   ```
   python3 -m venv graphrag-env
   ```

   Then, activate it:

   - On macOS/Linux:

     ```
     source graphrag-env/bin/activate
     ```

   - On Windows:

     ```
     .\graphrag-env\Scripts\activate
     ```

   You should see `(graphrag-env)` prefixed to your terminal prompt, indicating the environment is active.

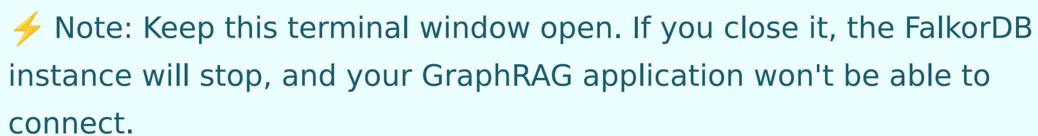
3. **Running FalkorDB Instance:** The FalkorDB GraphRAG SDK requires a running FalkorDB instance to store your knowledge graph. The easiest way to get one up and running for development is using Docker.

   First, ensure Docker is installed and running on your system. Then, pull and run the FalkorDB Docker image:

   ```
   docker run -p 6379:6379 -it --rm --name falkordb-instance falkordb/
   falkordb
   ```

   This command does a few things:

   - `docker run`: Starts a new container.
   - `-p 6379:6379`: Maps port 6379 on your host machine to port 6379 inside the container, which is where FalkorDB listens.
   - `-it`: Runs the container in interactive mode and allocates a pseudo-TTY.
   - `--rm`: Automatically removes the container when it exits.
   - `--name falkordb-instance`: Assigns a readable name to your container.
   - `falkordb/falkordb`: Specifies the Docker image to use.



   ⚡ Note: Keep this terminal window open. If you close it, the FalkorDB instance will stop, and your GraphRAG application won't be able to connect.

   You can verify FalkorDB is running by trying to connect to it with `redis-cli` (if you have it installed) in a new terminal:

   ```
   redis-cli
   127.0.0.1:6379> ping
   PONG
   ```

4. **LLM API Key:** The SDK needs an LLM to perform tasks like entity extraction and answer synthesis. We'll use OpenAI for this tutorial, but other LLMs (like Anthropic) are also supported.

   Set your OpenAI API key as an environment variable. This is the most secure way to handle sensitive keys in your code.

   - On macOS/Linux:

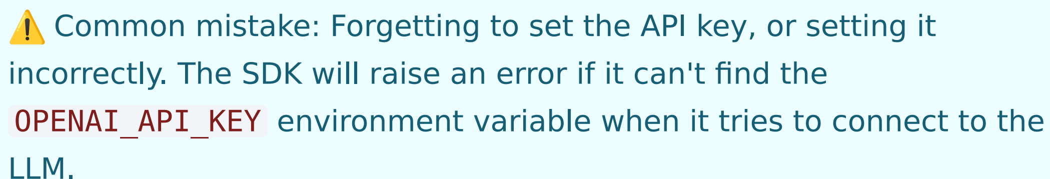
     ```
     export OPENAI_API_KEY="YOUR_OPENAI_API_KEY_HERE"
     ```

   - On Windows (Command Prompt):

     ```
     set OPENAI_API_KEY="YOUR_OPENAI_API_KEY_HERE"
     ```

   - On Windows (PowerShell):

     ```
     $env:OPENAI_API_KEY="YOUR_OPENAI_API_KEY_HERE"
     ```



> ⚠️ Common mistake: Forgetting to set the API key, or setting it incorrectly. The SDK will raise an error if it can't find the `OPENAI_API_KEY` environment variable when it tries to connect to the LLM.

What we've accomplished: You now have a dedicated Python environment, a running FalkorDB instance, and your LLM API key configured, ready for the SDK installation.

Installation of GraphRAG SDK

With your environment ready, installing the FalkorDB GraphRAG SDK is straightforward using pip.

1. **Install the SDK:** Make sure your virtual environment (`graphrag-env`) is still active. Then, install the SDK, specifying the exact version `1.0.0` for consistency with this tutorial.

   ```
   pip install graphrag-sdk==1.0.0
   ```

   This command will download and install the SDK and its dependencies.

2. **Verify the Installation:** You can quickly verify that the SDK is installed by trying to import it in a Python interpreter.

   Open a Python interpreter in your active virtual environment:

   ```
   python
   ```

   Then, try to import the `GraphRAG` class:

   ```
   from graphrag_sdk import GraphRAG
   print("FalkorDB GraphRAG SDK installed successfully!")
   exit()
   ```

   If you see the "FalkorDB GraphRAG SDK installed successfully!" message without any `ModuleNotFoundError`, you're good to go!

   What we've accomplished: You have successfully installed the FalkorDB GraphRAG SDK version 1.0.0 into your Python environment.

Core Concepts of GraphRAG

Understanding the core concepts behind GraphRAG will help you appreciate how the FalkorDB SDK works its magic. At its heart, GraphRAG combines the power of graph databases with the generative capabilities of LLMs.

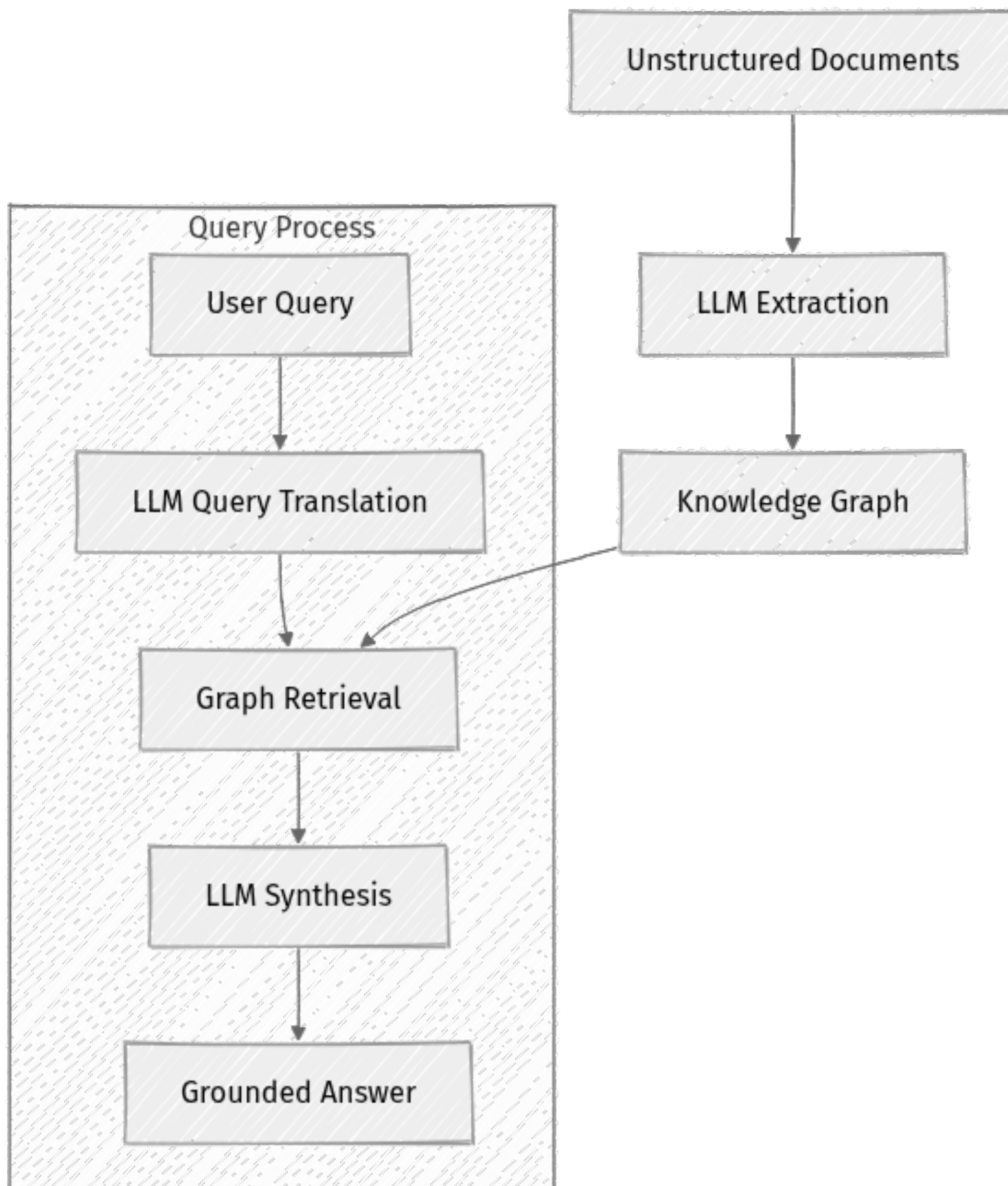
The GraphRAG Process:

1. **Document Ingestion:** You start with unstructured data – this could be text documents, articles, web pages, or even conversations.
2. **Knowledge Graph Construction:** The GraphRAG SDK, powered by an LLM, processes these documents. It identifies key **entities** (people, organizations, concepts, events) and the **relationships** between them. These entities and relationships are then stored in FalkorDB, forming a structured knowledge graph.
   - **Entities:** Nodes in the graph, representing distinct concepts.
   - **Relationships:** Edges between nodes, describing how entities are connected.
3. **Query Translation:** When a user asks a natural language question (e.g., "Who developed FalkorDB?"), the SDK doesn't just do a keyword search. Instead, it uses an LLM to translate this natural language question into a precise Cypher query (FalkorDB's graph query language).

4. **Graph Data Retrieval:** The generated Cypher query is executed against the FalkorDB knowledge graph. This retrieves not just individual facts, but a rich, interconnected subgraph of information that is highly relevant to the original question.
5. **LLM Synthesis:** The retrieved graph data (the "context") is then passed to the LLM along with the original natural language question. The LLM uses this context to synthesize a coherent, accurate, and grounded answer.

This structured retrieval process is what makes GraphRAG so powerful. By understanding the relationships between pieces of information, it can provide the LLM with a far more accurate and comprehensive context than simple keyword or vector similarity searches.

Here's a simplified flow of the GraphRAG process:



> 📌 Key Idea: GraphRAG transforms unstructured data into a structured knowledge graph, enabling precise, relationship-aware retrieval that significantly enhances LLM accuracy and reduces hallucinations.

What we've accomplished: You now have a solid understanding of how GraphRAG works, from document ingestion to generating a grounded answer, and the key role FalkorDB plays in this process.

Integrating with an LLM

The FalkorDB GraphRAG SDK relies on an LLM for several critical tasks, including extracting entities and relationships from documents, translating natural language queries into Cypher, and synthesizing final answers. Let's see how to configure the SDK to use your chosen LLM.

For this tutorial, we'll assume you're using OpenAI and have set your `OPENAI_API_KEY` environment variable as instructed in the "Prerequisites" section.

1. **Initialize the GraphRAG SDK with LLM Configuration:** The `GraphRAG` object needs to know which LLM to use. You specify this when you initialize the SDK. The SDK intelligently picks up the API key from the environment variable.

   Create a new Python file named `graphrag_app.py`.

   ```
   # graphrag_app.py
   import os
   from graphrag_sdk import GraphRAG

   # 🧠 Important: Ensure your OPENAI_API_KEY environment variable is set.
   # The SDK will automatically pick it up.

   try:
       # Initialize the GraphRAG SDK
       # We specify the LLM provider (e.g., "openai") and the model.
       # The SDK will look for OPENAI_API_KEY in environment variables.
       graphrag = GraphRAG(
           llm_provider="openai",
           llm_model="gpt-4o-mini" # You can use other models like "gpt-3.5-
   turbo" or "gpt-4"
       )
       print("GraphRAG SDK initialized successfully with OpenAI LLM.")
       print(f"Using LLM model: {graphrag.llm_model}")

   except Exception as e:
       print(f"Error initializing GraphRAG SDK: {e}")
       print("Please ensure your OPENAI_API_KEY environment variable is set
   correctly.")
   ```

   Let's break down the initialization:

   - `llm_provider="openai"`: Tells the SDK to use OpenAI's services.
   - `llm_model="gpt-4o-mini"`: Specifies the particular OpenAI model to use. `gpt-4o-mini` is a good choice for cost-effectiveness and performance for many RAG tasks. You can switch to `gpt-3.5-turbo` or `gpt-4` depending on your needs and budget.

2. **Run the script to verify LLM integration:**

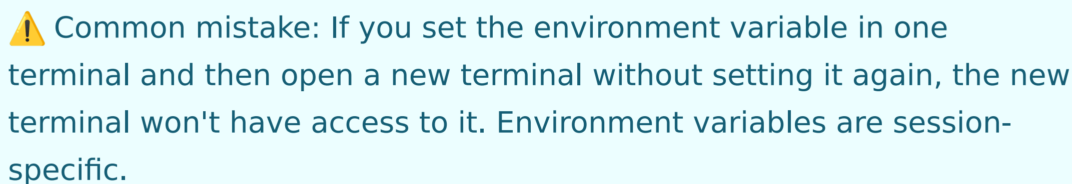
   Save the `graphrag_app.py` file and run it from your terminal (with your virtual environment active):

   ```
   python graphrag_app.py
   ```

   You should see output similar to this:

   ```
   GraphRAG SDK initialized successfully with OpenAI LLM.
   Using LLM model: gpt-4o-mini
   ```

   If you encounter an error, double-check that your `OPENAI_API_KEY` environment variable is correctly set and accessible in your current terminal session.

   What we've accomplished: You have successfully initialized the FalkorDB GraphRAG SDK, configuring it to connect with your OpenAI LLM using your API key. This is a crucial step for all subsequent operations.

Building a Knowledge Graph from Documents

Now for the exciting part: taking unstructured text and transforming it into a rich knowledge graph within FalkorDB. This is where the SDK truly shines, automatically extracting entities and their relationships.

For this example, we'll use a simple piece of text about FalkorDB and its origins.

1. **Prepare your document:** We'll embed the document directly into our Python script for simplicity. In a real-world application, you would load documents from files, databases, or APIs.

2. **Ingest the document into the Knowledge Graph:** We'll extend our `graphrag_app.py` file.

```
# graphrag_app.py
import os
from graphrag_sdk import GraphRAG
from falkordb import Graph

# ... (previous LLM initialization code) ...

# Define a sample document
sample_document = """
FalkorDB is an open-source graph database that is built on Redis.
It was originally developed by Redis Labs (now Redis) and later
spun off into its own independent project. FalkorDB is known for its
high performance and ability to handle large-scale graph data.
It supports the Cypher query language, making it accessible to
developers familiar with other graph databases. GraphRAG SDK
leverages FalkorDB for building intelligent GenAI applications.
"""

print("\n--- Ingesting Document ---")
try:
    # Ingest the document. The SDK will process it, extract entities and
relationships,
    # and store them in the FalkorDB instance connected to by the SDK.
    graphrag.ingest(sample_document)
    print("Document ingested successfully. Knowledge Graph updated.")

    # ⚡ Quick Note: The `ingest` method handles all the heavy lifting:
    # 1. Sending the document to the LLM for entity/relationship
extraction.
    # 2. Converting the extracted information into Cypher queries.
    # 3. Executing those queries against FalkorDB to build the graph.

except Exception as e:
    print(f"Error during document ingestion: {e}")
    print("Please check your FalkorDB instance and LLM connection.")
    exit()

# --- Verification ---
# Now, let's connect directly to FalkorDB to see what was created.
# The SDK creates a graph named 'graphrag'.
try:
    db_graph = Graph("graphrag")
    db_graph.query("MATCH (n) RETURN count(n) AS node_count")
    result = db_graph.query("MATCH (n) RETURN n.name AS entity_name,
labels(n) AS labels LIMIT 5")
    print("\n--- Verifying Knowledge Graph in FalkorDB (first 5 nodes)
---")
    if result.row_count > 0:
        for record in result.result_set:
            print(f"Entity: {record[0]}, Labels: {record[1]}")
    else:
        print("No nodes found in the graph. Ingestion might have
failed.")

    db_graph.query("MATCH ()-[r]->() RETURN count(r) AS
relationship_count")
    result_rels = db_graph.query("MATCH (a)-[r]->(b) RETURN a.name,
```

```
type(r), b.name LIMIT 5")
    print("\n--- Verifying Knowledge Graph Relationships (first 5) ---")
    if result_rels.row_count > 0:
        for record in result_rels.result_set:
            print(f"({record[0]})-[{record[1]}]->({record[2]})")
    else:
        print("No relationships found in the graph.")

except Exception as e:
    print(f"Error connecting to FalkorDB for verification: {e}")
    print("Ensure FalkorDB is running and accessible on port 6379.")
```

Explanation:

- `graphrag.ingest(sample_document)`: This single line orchestrates the entire process of parsing the document, extracting entities and relationships using the configured LLM, and storing them as nodes and edges in your FalkorDB instance.
- We then use `falkordb.Graph("graphrag")` to connect directly to the graph created by the SDK and run some simple Cypher queries (`MATCH (n) RETURN ...`) to inspect the nodes and relationships that were generated. The SDK uses the default graph name `graphrag`.

3. **Run the script to build the graph:**

   Save `graphrag_app.py` and run it:

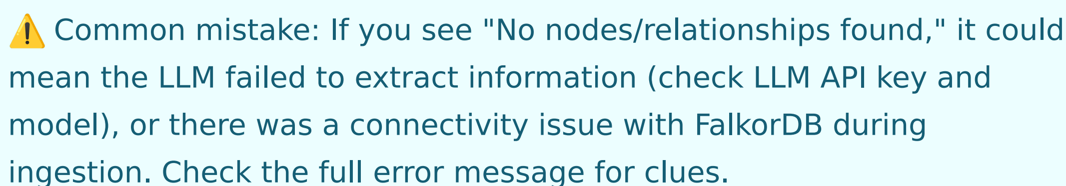
   ```
   python graphrag_app.py
   ```

   You will see output indicating the document ingestion and then the verification queries. The exact entities and relationships might vary slightly based on the LLM's interpretation, but you should see entries like:

   ```
   --- Ingesting Document ---
   Document ingested successfully. Knowledge Graph updated.

   --- Verifying Knowledge Graph in FalkorDB (first 5 nodes) ---
   Entity: FalkorDB, Labels: ['Database']
   Entity: Redis Labs, Labels: ['Organization']
   Entity: Redis, Labels: ['Organization']
   Entity: Cypher, Labels: ['Language']
   Entity: GraphRAG SDK, Labels: ['Software']

   --- Verifying Knowledge Graph Relationships (first 5) ---
   (FalkorDB)-[BUILT_ON]->(Redis)
   (FalkorDB)-[DEVELOPED_BY]->(Redis Labs)
   (Redis Labs)-[NOW_CALLED]->(Redis)
   (FalkorDB)-[SUPPORTS]->(Cypher)
   (GraphRAG SDK)-[LEVERAGES]->(FalkorDB)
   ```



What we've accomplished: You have successfully transformed an unstructured text document into a structured knowledge graph within FalkorDB using the GraphRAG SDK. You've also verified its contents directly using Cypher queries.

Performing a Basic GraphRAG Query

Now that our knowledge graph is populated, let's put the GraphRAG SDK to the test by asking a natural language question. The SDK will handle translating your question into a Cypher query, retrieving relevant data from FalkorDB, and synthesizing an answer using the LLM.

1. **Ask a natural language query:** We'll add a simple query to our `graphrag_app.py` script.

```
# graphrag_app.py
# ... (previous code for LLM initialization and document ingestion) ...

print("\n--- Performing Basic GraphRAG Query ---")
question = "Who developed FalkorDB and what is it built on?"

try:
    # Ask the question to the GraphRAG SDK
    response = graphrag.query(question)

    print(f"Question: {question}")
    print(f"Answer: {response}")

except Exception as e:
    print(f"Error during GraphRAG query: {e}")

print("Ensure your knowledge graph is populated and LLM connection is
active.")
```

Explanation:

- `graphrag.query(question)`: This is the core method for querying. Behind the scenes, the SDK:
    1. Sends your `question` to the LLM to generate a precise Cypher query.
    2. Executes that Cypher query against the FalkorDB knowledge graph to retrieve specific, relevant entities and relationships.
    3. Takes the retrieved graph context and the original `question`, sends them back to the LLM for final answer synthesis.

2. **Run the script to get an answer:**

   Save `graphrag_app.py` and run it:

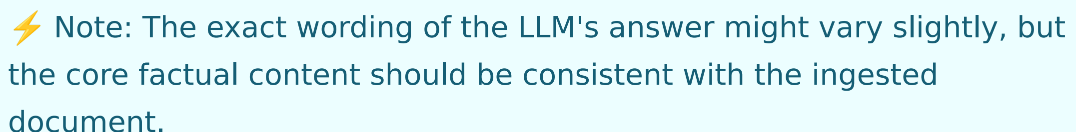
   ```
   python graphrag_app.py
   ```

   You should see the full output from setup, ingestion, verification, and finally, the answer to your query:

   ```
   ... (previous output) ...

   --- Performing Basic GraphRAG Query ---
   Question: Who developed FalkorDB and what is it built on?
   Answer: FalkorDB was originally developed by Redis Labs (now Redis) and
   is built on Redis.
   ```

   The answer is directly derived from the information we ingested into our knowledge graph, demonstrating how GraphRAG provides grounded, accurate responses.

   > ⚡ Note: The exact wording of the LLM's answer might vary slightly, but the core factual content should be consistent with the ingested document.

   What we've accomplished: You have successfully performed a natural language query against your FalkorDB knowledge graph using the GraphRAG SDK, receiving a contextually accurate answer.

Performance Benefits and Practical Use Cases

The FalkorDB GraphRAG SDK 1.0 isn't just about accuracy; it also brings significant performance advantages and opens up a wide array of practical use cases for developers.

Performance Benefits:

1. **Reduced Token Costs:** By precisely retrieving only the most relevant subgraph of information, GraphRAG minimizes the amount of data sent to the LLM as context. This directly translates to lower API costs, especially with larger documents or complex queries.

2. **Faster Response Times:** Efficient graph traversal in FalkorDB allows for quicker retrieval of context compared to broad vector searches or keyword matching across large datasets. This leads to faster end-user response times for your GenAI applications.
3. **Enhanced Accuracy and Reduced Hallucinations:** This is the primary driver. By feeding the LLM with structured, interconnected facts, the likelihood of the LLM generating incorrect or irrelevant information drastically decreases. The GraphRAG SDK has been shown to outperform other frameworks on benchmarks like GraphRAG-Bench, specifically because of its superior retrieval layer.
4. **Scalability:** FalkorDB, being built on Redis, inherits its high performance and scalability, allowing your knowledge graphs to grow to massive sizes without sacrificing query speed.

Practical Use Cases:

- **Intelligent Q&A over Complex Documents:** Imagine having to answer questions from thousands of pages of legal documents, engineering specifications, or medical research. GraphRAG can build a knowledge graph from these, allowing users to ask precise questions and get grounded answers, even when information is scattered across documents.
- **Domain-Specific Chatbots:** Build chatbots that act as experts in a particular domain (e.g., customer support for a specific product, internal knowledge base assistants). The graph ensures the chatbot stays "on topic" and provides accurate information specific to its domain.
- **Personalized Recommendations:** By modeling user preferences, product attributes, and interactions as a graph, GraphRAG can power highly personalized recommendation engines that understand subtle connections.
- **Fraud Detection:** Graph analysis is powerful for detecting patterns of fraud. GraphRAG can help explain why a transaction or entity is flagged by querying the underlying graph of connections and activities.
- **Supply Chain Optimization:** Model complex supply chains as a graph to identify bottlenecks, optimize routes, or predict disruptions based on interconnected events and entities.

> ⚡ Real-world insight: Companies often struggle with LLMs providing generic or "confidently wrong" answers. GraphRAG addresses this by making the retrieval layer highly intelligent, ensuring the LLM always gets the best possible context, leading to more trustworthy and actionable insights.

What we've accomplished: You now understand the significant performance and accuracy benefits of using FalkorDB GraphRAG SDK, along with a range of practical scenarios where it can be applied to build powerful GenAI applications.

Common Issues and Troubleshooting

Even with the best tools, you might encounter issues. Here are some common problems when getting started with the FalkorDB GraphRAG SDK and how to troubleshoot them.

1. **FalkorDB Instance Not Running or Accessible:**
    - **Symptom:** `ConnectionError`, `RedisConnectionError`, or the SDK hangs during `ingest` or `query`.
    - **Troubleshooting:**
        - **Check Docker:** Ensure your FalkorDB Docker container is running. In a new terminal, run `docker ps`. You should see `falkordb-instance` listed. If not, restart it using `docker run -p 6379:6379 -it --rm --name falkordb-instance falkordb/falkordb`.
        - **Port Conflict:** Make sure no other service is using port 6379 on your machine.
        - **Firewall:** Check if a firewall is blocking access to port 6379.

2. **Missing or Incorrect LLM API Key:**

    - **Symptom:** Errors related to `AuthenticationError`, `InvalidAPIKey`, or `EnvironmentVariableError` from the LLM provider (e.g., OpenAI). The SDK might also explicitly state it cannot find `OPENAI_API_KEY`.
    - **Troubleshooting:**
        - **Verify Environment Variable:** In the same terminal where you run your Python script, type `echo $OPENAI_API_KEY` (Linux/macOS) or `echo %OPENAI_API_KEY%` (Windows Cmd) / `echo $env:OPENAI_API_KEY` (Windows PowerShell). Ensure it prints your actual API key.
        - **Re-export:** If you opened a new terminal or restarted your computer, environment variables might need to be re-exported.
        - **Key Validity:** Double-check your API key on the LLM provider's website (e.g., OpenAI dashboard) to ensure it's active and not expired.

3. **Python Environment Issues (Dependencies):**

    - **Symptom:** `ModuleNotFoundError: No module named 'graphrag_sdk'` or other dependency-related errors.
    - **Troubleshooting:**
        - **Activate Virtual Environment:** Ensure your `graphrag-env` virtual environment is active before running `pip install` or your Python script. You should see `(graphrag-env)` in your terminal prompt.
        - **Install Correctly:** Re-run `pip install graphrag-sdk==1.0.0` to ensure all dependencies are installed.
        - **Python Version:** Confirm you are using Python 3.10+ by running `python3 --version`.

4. **Graph Generation Errors (Ingestion Issues):**
    - **Symptom:** `Errors not properly defined` or `cannot generate an ontology from my markdown txt file` (as seen in GitHub issues), or the verification step shows "No nodes found in the graph."
    - **Troubleshooting:**
        - **LLM Model Performance:** Sometimes, for very short or ambiguous texts, the LLM might struggle to extract entities and relationships. Try a slightly longer or clearer document.
        - **LLM Rate Limits:** If you're ingesting many documents quickly, you might hit LLM rate limits. The SDK usually handles retries, but persistent issues might require checking your LLM provider's usage dashboard.
        - **SDK Logs:** The SDK might provide more detailed logs if you configure a logger. For debugging, you can add `logging.basicConfig(level=logging.DEBUG)` to your script.
        - **Document Complexity:** Start with simple, factual documents. Overly complex or poorly formatted text can be challenging for initial entity extraction.
5. **Poor Query Results (Irrelevant or Incomplete Answers):**
    - **Symptom:** The LLM provides a generic answer, says it doesn't know, or gives an answer not grounded in your ingested data.
    - **Troubleshooting:**
        - **Knowledge Graph Content:** Is the information needed to answer the question actually present in your ingested document and subsequently in the graph? Verify this with direct Cypher queries to FalkorDB.
        - **Query Ambiguity:** Is your natural language question clear and specific enough? Ambiguous questions can lead to ambiguous Cypher queries and less precise retrieval.
        - **LLM Model Choice:** A less capable LLM model (e.g., an older `gpt-3.5-turbo` version) might struggle with complex query translation or answer synthesis. Consider trying a more powerful model like `gpt-4o-mini` or `gpt-4`.
        - **Graph Schema:** While the SDK handles schema generation, understanding the types of nodes and relationships created can help you formulate better questions.

What we've accomplished: You are now equipped with common troubleshooting steps to resolve issues you might encounter when developing with the FalkorDB GraphRAG SDK.

What to Build Next

Congratulations on building your first GraphRAG application with FalkorDB! This is just the beginning. Here are three concrete ideas to extend your project and explore more advanced capabilities:

1. **Ingest Multiple Documents from a Directory:** Instead of a single hardcoded string, modify your `graphrag_app.py` to read and ingest multiple text files from a specified directory. This will allow you to build a more comprehensive knowledge graph. You could create a `data/` folder, add a few `.txt` files with information about different topics (e.g., company history, product features, famous people), and then iterate through them, calling `graphrag.ingest()` for each file. This will demonstrate how GraphRAG scales with more data.
2. **Implement a Simple Command-Line Chatbot:** Turn your basic query into an interactive experience. Create a loop where the user can continuously type questions, and your GraphRAG application provides answers. This would simulate a simple chatbot. You could even add a "quit" command to exit the loop. This helps solidify the query flow and makes your application more engaging.

```
# Example snippet for interactive query loop
# ... (after graphrag initialization and ingestion) ...
print("\n--- Start Interactive Q&A (type 'quit' to exit) ---")
while True:
    user_question = input("Your question: ")
    if user_question.lower() == 'quit':
        break
    response = graphrag.query(user_question)
    print(f"Answer: {response}\n")
print("Exiting Q&A.")
```

3. **Explore Graph Visualization:** After ingesting more data, the knowledge graph in FalkorDB will become quite rich. Connect to your FalkorDB instance using a graph visualization tool like FalkorDB Browser or Graph Data Science Library (GDS) (if you install it and use its visualization features). Run Cypher queries to retrieve subgraphs related to your questions and visualize them. Seeing the entities and relationships visually will provide a deeper understanding of how the SDK constructs its knowledge base and retrieves context. This can be incredibly insightful for debugging and understanding your data.