Graph RAG Documentation

What is Graph RAG?

Graph RAG (Graph-based Retrieval-Augmented Generation) is an advanced evolution of traditional Retrieval-Augmented Generation (RAG) that leverages knowledge graphs to enhance large language models (LLMs) with structured, relational context. Unlike standard RAG—which typically retrieves relevant text chunks based solely on vector similarity—Graph RAG builds a network of entities (nodes) and relationships (edges) that can support multi-hop reasoning and offer holistic data understanding. This makes it particularly useful for complex queries that require reasoning across interconnected concepts, such as “What influenced the influencers of modern physics?” or for summarizing themes in large datasets like books.

The Process, Step-by-Step

Here’s how Graph RAG typically works:

1. **Knowledge Graph Construction**
    - **Data Ingestion**: Start with raw data—text corpora, databases, or documents. This could be anything from Wikipedia articles to research papers or even social media posts (like from X, if we wanted to get creative!).
    - **Entity Extraction**: Use natural language processing (NLP) techniques, like named entity recognition (NER), to identify key entities (e.g., people, places, concepts).
    - **Relationship Extraction**: Determine how these entities are connected (e.g., "Elon Musk" → "founded" → "xAI"). This often involves dependency parsing or pre-trained relation extraction models.
    - **Graph Building**: Store the entities as nodes and relationships as edges in a graph database (e.g., Neo4j) or an in-memory graph structure. Add attributes to nodes and edges for extra context (e.g., dates, weights).
2. **Query Processing**
    - When you ask a question (e.g., "How does Graph RAG improve over traditional RAG?"), the system first interprets it using NLP to identify key entities and intent.
    - The query is mapped to the knowledge graph—finding relevant nodes (like "Graph RAG" and "traditional RAG") and their relationships.
3. **Retrieval with Graph Traversal**
    - Unlike traditional RAG, which might rely on a vector search over unstructured text, Graph RAG traverses the graph to retrieve connected information.

    - For example, it might follow edges like "Graph RAG" → "uses" → "knowledge graph" → "enhances" → "contextual retrieval" to gather a subgraph of relevant data.
    - This subgraph provides structured context—more precise and relationally rich than a flat list of documents.
4. **Augmentation**
    - The retrieved subgraph is converted into a format the language model can use, like a textual summary or a set of triples (e.g., "Graph RAG improves accuracy by leveraging relationships").
    - This augmented context is fed into the generation model alongside your original query.
5. **Generation**
    - The model (like me!) generates a response using both its internal knowledge and the graph-augmented context. The result is more grounded in structured data, reducing hallucinations and improving relevance.

Methods in Graph RAG

1. Knowledge Graph Construction

- **Entity & Relationship Extraction:**
    - **Method:** Utilize an LLM or specialized NLP tools (like spaCy or OpenIE) to identify entities and relationships.
    - **Output:** Triples that represent connections between entities.
- **Graph Building:**
    - **Method:** Store the extracted triples as nodes and edges in a graph database.
    - **Detail:** Nodes represent entities; edges represent the relationships, forming an interconnected network.

2. Indexing for Retrieval

- **Vector Embeddings:**
    - **Method:** Use models (e.g., all-MiniLM-L6-v2) to convert node text or triples into vector embeddings.
    - **Purpose:** Facilitate fast similarity-based searches.
- **Graph-Specific Indexing:**
    - **Method:** Leverage graph algorithms (e.g., PageRank, community detection) to identify influential nodes or clusters.
    - **Purpose:** Enhance retrieval by highlighting key entities and their groups.

3. Retrieval Methods

- **Graph Traversal:**
    - **Method:** Follow edges starting from a queried node to explore related nodes.
    - **Tool:** Execute Cypher queries (in Neo4j) or use custom traversal logic.
- **Vector Search:**
    - **Method:** Compare query embeddings to node embeddings to retrieve the top-k relevant nodes.
    - **Tool:** Use vector indexes (e.g., Neo4j Vector, FAISS) for efficient search.
- **Hybrid Retrieval:**
    - **Method:** Combine vector-based retrieval (to quickly identify candidate nodes) with graph traversal (to capture relational depth).
    - **Purpose:** Achieve a balance between retrieval speed and relational context.

4. Context Augmentation

- **Subgraph Extraction:**
    - **Method:** Extract a connected subset of the graph around the relevant nodes (e.g., nodes within two hops).
    - **Usage:** Serialize the subgraph into text that the LLM can understand.
- **Community Summaries:**
    - **Method:** Cluster related nodes using community detection algorithms and then summarize these clusters.
    - **Purpose:** Provide a higher-level overview for broad queries.

5. Generation

- **Prompt Engineering:**
    - **Method:** Construct a prompt that includes both the query and the retrieved structured context.
    - **Result:** The LLM produces a natural language answer that is grounded in the graph's structured data.

Frameworks for Graph RAG

LangChain

- **Description:**
  A modular framework for LLM applications with strong support for both RAG and graph-based workflows.
- **Key Components:**
    - **Neo4jGraph:** Connects to Neo4j for graph storage and querying.
    - **GraphQAChain:** Enables graph-based question-answering using Cypher queries.

    - **Neo4jVector:** Creates a vector index for hybrid retrieval approaches.

LlamaIndex

- **Description:**
  Focused on RAG with robust graph integration.
- **Key Components:**
    - **KnowledgeGraphIndex:** Builds and indexes knowledge graphs.
    - **KnowledgeGraphRAGQueryEngine:** Combines vector and graph retrieval seamlessly.

Microsoft GraphRAG

- **Description:**
  An open-source system developed by Microsoft Research that includes advanced features such as Query-Focused Summarization (QFS).
- **Considerations:**
    - Resource-intensive and may require tuning for out-of-the-box use.

Haystack (Deepset)

- **Description:**
  A modular NLP framework that supports Graph RAG workflows with components for both vector search and graph-based retrieval.

Tools for Graph RAG

Graph Databases

- **Neo4j:**
    - **Description:** Leading graph database supporting Cypher queries and vector indexing.
    - **Pricing (as of early 2025):**
        - **Aura Free:** Free tier with limited resources.
        - **Aura Professional:** Starts at **$65/month**.
        - **Aura Enterprise:** Custom pricing based on workload and support requirements.
- **Memgraph:**
    - **Description:** An in-memory graph database optimized for real-time applications.
    - **Pricing (as of early 2025):**

        - **Community Edition:** Free.
        - **Memgraph Cloud:** Offers a free tier; paid plans start at approximately **$0.08 per hour**. For continuous 24/7 deployment, this is roughly **$58–$60/month**. Enterprise plans and higher-resource deployments are available on a custom basis.
- **NetworkX:**
    - **Description:** A Python library for building in-memory graphs, best suited for smaller datasets.
    - **Pricing:** Free and open-source under the BSD license.

LLMs

- **Ollama:**
  Supports local models like Llama 3.1 for tasks such as triple extraction and response generation.
- **OpenAI API:**
  Utilize models like GPT-3.5 or GPT-4 for cloud-based generation.
    - **Pricing:** Refer to the OpenAI Pricing page for the latest rates, which typically charge per token.

Embedding Models

- **Sentence-Transformers:**
  Models like all-MiniLM-L6-v2 provide pre-trained embeddings for similarity search.

Data Processing Tools

- **PyPDF2:**
  Extracts text from PDFs.
- **spaCy:**
  Alternative for entity and relationship extraction.

Vector Stores (Optional)

- **FAISS:**
  Fast in-memory vector search.
- **Milvus:**
  Scalable vector database for larger deployments.

Implementation Steps

1. **Setup:**
   - Install and configure necessary tools: Neo4j, the chosen LLM provider (OpenAI or Ollama), and relevant Python packages.
   - Ensure your environment is configured with the necessary API keys (e.g., OpenAI).
2. **Data Ingestion:**
   - Extract raw text from sources (e.g., PDFs) using tools like PyPDF2.
3. **Knowledge Graph Construction:**
   - Extract entity-relation triples using an LLM (via prompt-based extraction) or alternative NLP tools.
   - Load these triples into your graph database (e.g., using Neo4j).
4. **Indexing:**
   - Compute vector embeddings for nodes using models such as Sentence-Transformers.
   - Build a vector index (via tools like Neo4jVector or FAISS) to support hybrid retrieval.
5. **Query Processing:**
   - Map user queries to the graph by retrieving relevant nodes and their relationships.
   - Augment the query with the retrieved context (either as a subgraph or a summary).
6. **Response Generation:**
   - Feed the augmented prompt to your LLM to generate a final, natural language response.
7. **Output:**
   - Return the generated answer for display or further processing.

Challenges and Tips

- **Parsing:**
  Triple extraction can be noisy. Ensure robust parsing mechanisms (e.g., fallback regex or validation) are in place.
- **Scalability:**
  In-memory graphs work well for small datasets, but a full-scale solution will benefit from a dedicated graph database like Neo4j.

- **Performance:**
  Large models like Llama 3.1 require significant resources (e.g., RAM). Adjust parameters and consider using cloud-based APIs if local resources are limited.
- **Error Handling:**
  Incorporate error handling and logging throughout the pipeline—from data ingestion to final generation—to streamline debugging and ensure reliability.

Resources

- **LangChain Documentation:**
  Refer to the LangChain GitHub repository and its community modules for graph and vector integration.
- **Neo4j GraphRAG Guides and Pricing:**
  Explore Neo4j's official blog and pricing page for advanced graph-based retrieval techniques and up-to-date pricing.
- **Memgraph:**
  Visit the Memgraph website for the latest pricing details.
- **Microsoft GraphRAG:**
  Visit the Microsoft GraphRAG GitHub page for additional insights.
- **NetworkX:**
  Learn more about NetworkX on its official website; it is free and open-source.
- **OpenAI Pricing:**
  Detailed pricing is available on the OpenAI Pricing page.