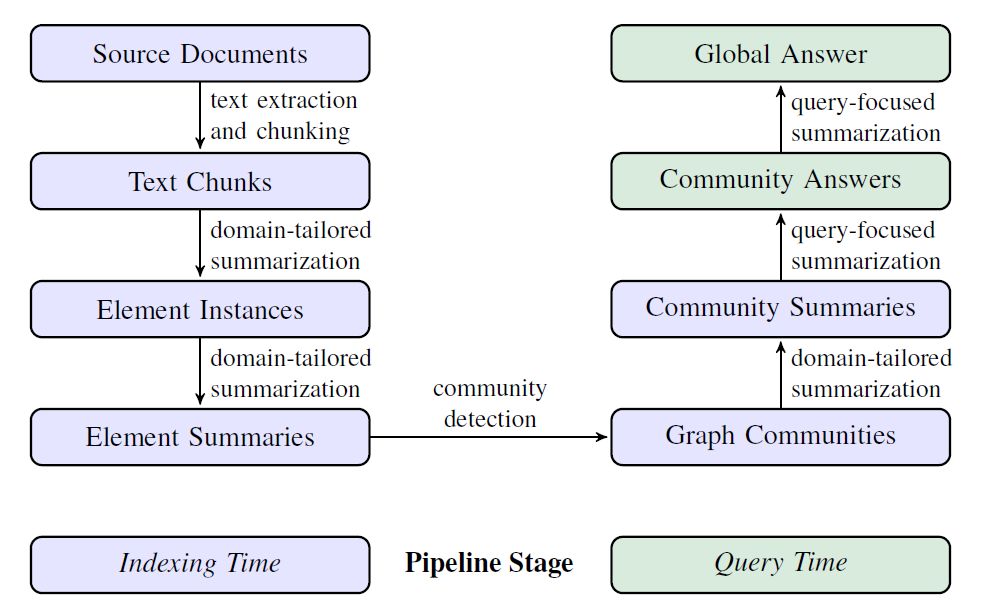
**From Local to Global: A Graph RAG Approach to**

**Query-Focused Summarization**

Graph RAG approach uses the natural modularity of graphs to partition data for global summarization. It uses an LLM to build a graph-based text index in two stages:

1. Derive an entity knowledge graph from the source documents,
2. Pre-generate community summaries for all groups of closely related entities.

Can answer such questions like “What are the main themes in the dataset?”, Basically, it is an inherently query focused summarization (QFS) task. Graph RAG approach improves the question answering over private text corpora that scales with both the generality of user questions and the quantity of source text to be indexed. Graph RAG leads to substantial improvements for both the comprehensiveness and diversity of generated answers.



**Figure 1: Graph RAG pipeline using an LLM-derived graph index of source document text**

**Data Injection:**

1. *Documents/chunks/Text Preprocessing:*
2. To reduce document size and improve latency use text summarization for heavy documents or multi-documents with the below steps:

Step1: LLM (use a specific LLM embedding to summarize documents)

Step2: Knowledge graph to reduce size with entities, relationship and their properties with subgraphs.

Note: Above steps can be followed bidirectionally.

1. Create vector DB/Embedding/Indexing with LLM embedding.

**Storing vector Embedding/Indexing**

* Generate KG with the embedding and store in graph DB or store the embedding in FAISS/pinecone to improve latency and accuracy.

or

Both the methods can be combined (KG+ vector embedding) and store DB to handle both structure and unstructured data

* Generate four communities (C0,C1,C2,C3) Graph RAG summary from the embedding/KG of the document/multi-documents/embeddings by using text summarized Map Reduced approach.
* **C0:** Uses root-level community summaries (fewest in number) to answer user queries.
* **C1:** Uses high-level community summaries to answer queries. These are sub-communities.

of C0, if present, otherwise C0 communities projected down.

* **C2:** Uses intermediate-level community summaries to answer queries. These are subcommunities of C1, if present, otherwise C1 communities projected down.
* **C3:** Uses low-level community summaries (greatest in number) to answer queries. These

are sub-communities of C2, if present, otherwise C2 communities projected down.

**A diagram of a network

Description automatically generated**

C0

C3

C1

C2

**Figure 2.1 Communities’ Summary Figure.2.2 Communities Graph**

**Chat Response/Architecture:**

Multi-hope RAG, memory-based response, Head-to-Head measures.

Head-To-Head measures can compute using an LLM evaluator are as follows:

• *Comprehensiveness*: How much detail does the answer provide to cover all aspects and

details of the question?

• *Diversity:* How varied and rich is the answer in providing different perspectives and insights on the question?

**Proposed Architecture**

A blue circle with a person in it

Description automatically generated

**Response**

**LLMRAG/GraphRAG**

**(Response augmentation)**

**Final combine Context + past response**

**Past response history**

User Query

A white background with blue text

Description automatically generated

1. *Entity knowledge graph from the source documents,*
2. *Pre-generate community summaries for all groups of closely related entities (KNN/cosine similarity/BERT/RoBERT).*

Read response.

**Chatbot Architecture**

Chunks

Input Documents/Text

A stack of papers with text

Description automatically generated

A group of black and white icons

Description automatically generated



Memory

A white brain with circuit lines

Description automatically generated with medium confidence

User

Partial Context retrieval

**Partial context generated by each community response history.**

Augment response.

Similarity search

**LLM Text embedding /KG for each community summary**

**Indexed Vector DB (pinecone) / KCG (Knowledge Community (C0, C1, C2, C3) Graph in 2stages)**

Generate response.

Write response

**User Query**