**From Local to Global V1.1**

*“” Paper1: From Local to Global: A Graph RAG Approach to Query-Focused Summarization””*

1. Graph RAG Approach & Pipeline (Figure1\_v1):

1. Source document- -> Text chunks (token size: 600-2400)

2. Text Chunks-🡪 Elements Instances:

* To domain the document for context learning, use multipart LLM prompt to Identify and extract instances for the graph nodes and edges including source and target from each chunk of source document.

1. Text chunk 🡪 Multilevel LLM prompt (to domain the document for context learning) 🡪find all entities (name, type, description (including source & target)) 🡪 identify instances for nodes & edges.
2. Abstractive Summary/Generate tuple ((source/object entities, name, type, description of entities/relationships/claims), (all entities, name, type, description of entities/relationships/claims))

Note: It also supports a secondary extraction prompt for any additional covariates associated with the extracted node instances. It extracts the claim linked to detected entities, including the subject, object, type, description, source text span, and start and end dates. And make sure no entities are left in multistage LLM prompt by asking YES (many entities are left)/NO.

3.Element instances 🡪 Element Summaries:

* Element instances/Abstractive Summary(tuples)🡪 LLM to summarize 🡪Semantic instance level Element Summary
* To convert all such instance-level summaries into single blocks of descriptive text for each graph element (i.e., entity node, relationship edge, and claim covariate) requires a further round of LLM summarization over matching groups of instances. i.e.

Instance-level-summary🡪LLM Vector embedding 🡪 KNN/cosine similarity search to find homogeneous summary clusters 🡪 LLM to summarize homogeneous clusters 🡪 single block summary of similar instances Elements’/homogenous clusters’ summary.

Note: potential concern at this stage is that the LLM may not consistently extract references to the same entity in the same text format, resulting in duplicate entity elements and thus duplicate nodes in the entity graph. However, since all closely-related “communities” of entities will be detected and summarized in the following step, and given that LLMs can understand the common entity. There should be sufficient connectivity from all variations to a shared set of closely-related entities.

4. Element summaries🡪Graph Communities:

1. Indexed Element summary of homogeneous clusters🡪 neo4j🡪 Homogeneous weighted undirected Graph
2. Homogeneous weighted undirected Graph🡪 Graph Community Detection algorithms (Hierarchical community structure) 🡪Partition graph into communities of nodes

Note: To recover hierarchical community structure of large-scale graphs efficiently. Each level of this hierarchy provides a community partition that covers the nodes of the graph in a mutually-exclusive, collective-exhaustive way, enabling divide-and-conquer global summarization.

5. Graph Communities🡪 Communities Summaries:

* + Graph communities🡪Lieden Hierarchy method 🡪 community summaries (on global summarized graph summaries

Graph based communities are used to generate Communities’ summaries. These communities’ summaries are independently useful to understand the global structure and semantics of the dataset and may themselves be used to make sense of a corpus in the absence of a question. For example, a user may scan through community summaries at one level looking for general themes of interest, then follow links to the reports at the lower level that provide more details for each of the subtopics.

* *Leaf-level communities*. The element summaries of a leaf-level community (nodes, edges, covariates) are prioritized and then iteratively added to the LLM context window until the token limit is reached. The prioritization is as follows: for each community edge in decreasing order of combined source and target node degree (i.e., overall prominence), add descriptions of the source node, target node, linked covariates, and the edge itself.
* Higher-level communities. If all element summaries fit within the token limit of the context window, proceed as for leaf-level communities, and summarize all element summaries within the community. Otherwise, rank sub-communities in decreasing order of element summary tokens and iteratively substitute sub-community summaries (shorter) for their associated element summaries (longer) until fit within the context window is achieved.

6. Community Summaries 🡪Community Answers🡪Global Answers:

1. For a given community level, the global answer to any user query is generated as follows:

Divide randomly shuffled Community summary into chunks (community summaries preparation) 🡪 parallelly generates answers from each chunk (Answer mapping) 🡪 Reduce to global answer.

* + *Prepare community summaries*: community summaries are randomly shuffled and divided into chunks of pre-specified token size. This ensures relevant information is distributed across chunks, rather than concentrated (and potentially lost) in a single context window.
  + *Map community answers:* Generate intermediate answers in parallel, one for each chunk. The LLM is also asked to generate a score between 0-100 indicating how helpful the generated answer is in answering the target question. Answers with score 0 are filtered out.
  + *Reduce to global answer:* Intermediate community answers are sorted in descending order of helpfulness score and iteratively added into a new context window until the token limit is reached. This final context is used to generate the global answer returned to the user.
* **Communities Comparison**

Six conditions can compare six, including Graph RAG using four levels of graph communities (C0, C1, C2, C3), a text summarization method applying our map-reduce approach directly to source texts (TS), and a naive “semantic search” RAG approach (SS):

1. CO: Uses root-level community summaries (fewest in number) to answer user queries.
2. C1: Uses high-level community summaries to answer queries. These are sub-communities of C0, if present, otherwise C0 communities projected down.
3. C2: Uses intermediate-level community summaries to answer queries. These are subcommunities of C1, if present, otherwise C1 communities projected down.
4. 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.
5. TS: Except source texts (rather than community summaries) are shuffled and chunked for the map-reduce summarization stages.
6. SS: An implementation of naive RAG in which text chunks are retrieved and added to the available context window until the specified token limit is reached.

The size of the context window and the prompts used for answer generation are the same across all six conditions (except for minor modifications to reference styles to match the types of contexts.

***Conclusion***: **A Trade-offs of building a graph index achieves the best head-to-head results against other methods, but in many cases the graph-free approach to global summarization of source texts performed competitively. The real-world decision about whether to invest in building a graph index depends on multiple factors, including the compute budget, expected number of lifetime queries per dataset, and value obtained from other aspects of the graph index (including the generic communities’ summaries and the use of other graph-related RAG approaches).**