



REDFIELD



MEM
GRAPH

How Would Microsoft GraphRAG Work Alongside Actual Graph Database?

Memgraph + Redfield - 2025-02-18





Welcome!

- Consultancy based in Stockholm, Sweden
- Expertise in:
 - Graph Analytics
 - Machine Learning
 - Data Science
 - RAG Applications
 - Custom Chatbots
 - API Development
- Build custom solutions
- Training programs



Jacob
Coles



Redfield
redfield.ai



Today's Goal

- Explore and compare Retrieval-Augmented Generation (RAG) techniques
 - Naive RAG approach
 - vs
 - GraphRAG approach with global-search
- How Memgraph 3.0's new features made this possible:
 - Vector Search
 - Leiden Community Detection Algorithm
- Motivated by Microsoft's GraphRAG approach: How to build a GraphRAG system with Q&A functionality that has global knowledge over the entire graph



Jacob
Coles

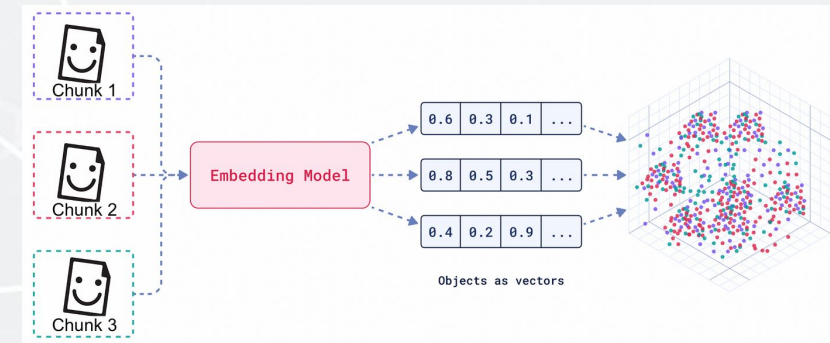


Redfield
redfield.ai

Introduction: Naive RAG (Retrieval-Augmented Generation)

'Regular' RAG Utilising a Large Language Model (LLM)

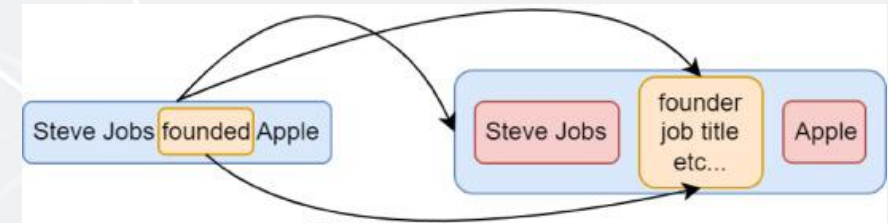
- RAG usually involves a simple document search, where relevant information is pasted into the context (i.e. prompt) of the LLM
- The indexing process:
 - Each document is split into 'Chunks': splitting documents into manageable components
 - Embeddings are generated from the chunks (also using an LLM)
 - The embeddings and the corresponding text are stored in a vector-index
- The querying process:
 - A user's question is turned into an embedding, which is used to then search for 'Chunks' in the documents
 - Relies on a vector search of semantically similar text content
 - These chunks are retrieved as text, and provided to the LLM as context for the user's question
 - The LLM will answer the question by considering the additional information provided by the document chunks



GraphRAG (Graph Retrieval-Augmented Generation)

Built To Be More Connected

- GraphRAG doesn't have one fixed definition but this is what people usually mean: Using LLM to transform 'flat' text into a graph, in order to have more organised and connected data. As a consequence:
 - GraphRAG allows entity-level search (whereas naive or 'regular' RAG can only fetch document chunks)
 - Graph traversal/search
- To perform this graph-generation or 'indexing':
 - We read in text-documents
 - Extract the triplet of 'subject', 'relation' and 'object' using an LLM
 - Resolve common references to subjects and objects (entity resolution)
 - Upload to the graph database
- For searching the graph:
 - Use a graph analytics tool (like Memgraph Labs), or graph query language (cypher)
 - Generating graph query / cypher query using an LLM
 - Vector-search over entities (optionally combined with graph traversal)



Problems With Naive GraphRAG

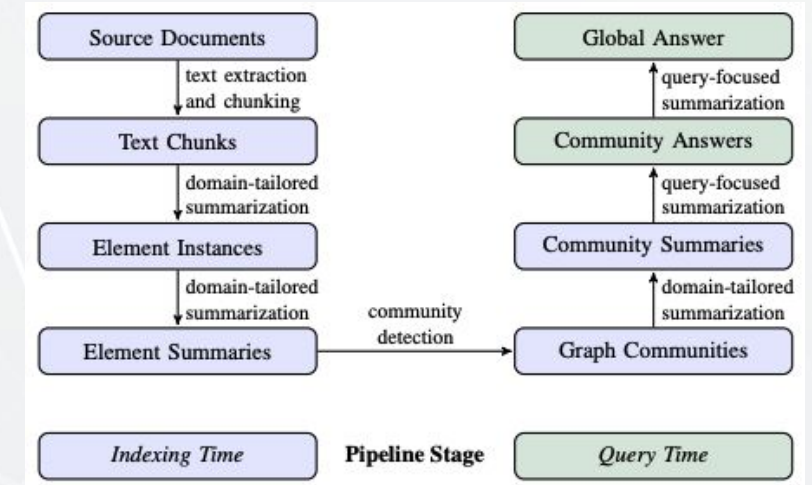
What Are The Limitations?

- Knowledge is still limited to specific localities:
 - We can only fetch specific entities or document chunks
 - Large-scale or ‘global’ generalisations aren’t always made
 - Long-distance dependencies are often missed (without extensive graph traversal)
- Many of the same limitations as regular document-based RAG such as:
 - Limited context window of the LLM
 - Accuracy limitations across vast amounts of data
 - Connecting disparate concepts are almost impossible
 - Queries like as “What are the top 5 themes in the book?” perform poorly since a vector search looks for semantically similar text content within specific pieces of text

The Solution

Microsoft GraphRAG Implementation (With Global Search)

- In 2024 Microsoft released a new concept to bridge the gap of global knowledge:
 - D. Edge et al. “From Local to Global: A Graph RAG Approach to Query-Focused Summarization”. Microsoft Research. Retrieved from <https://arxiv.org/pdf/2404.16130>
- Builds on top of previous GraphRAG-architecture
- Utilises Leiden community-detection to group well-connected pieces of knowledge
 - Conceptually, these communities or ‘groupings’ represent a higher-level groupings of related nodes
 - The communities are structured hierarchically on multiple levels to refer to different layers of knowledge.
 - A low-level community may refer to a few specific entities
 - A high-level community links to other lower-level communities and act as a bridge to large portions of the graph
- Community-summaries are attached to the community nodes, and embeddings are created
- Communities can be searched over (using embeddings) to find information summaries and global information



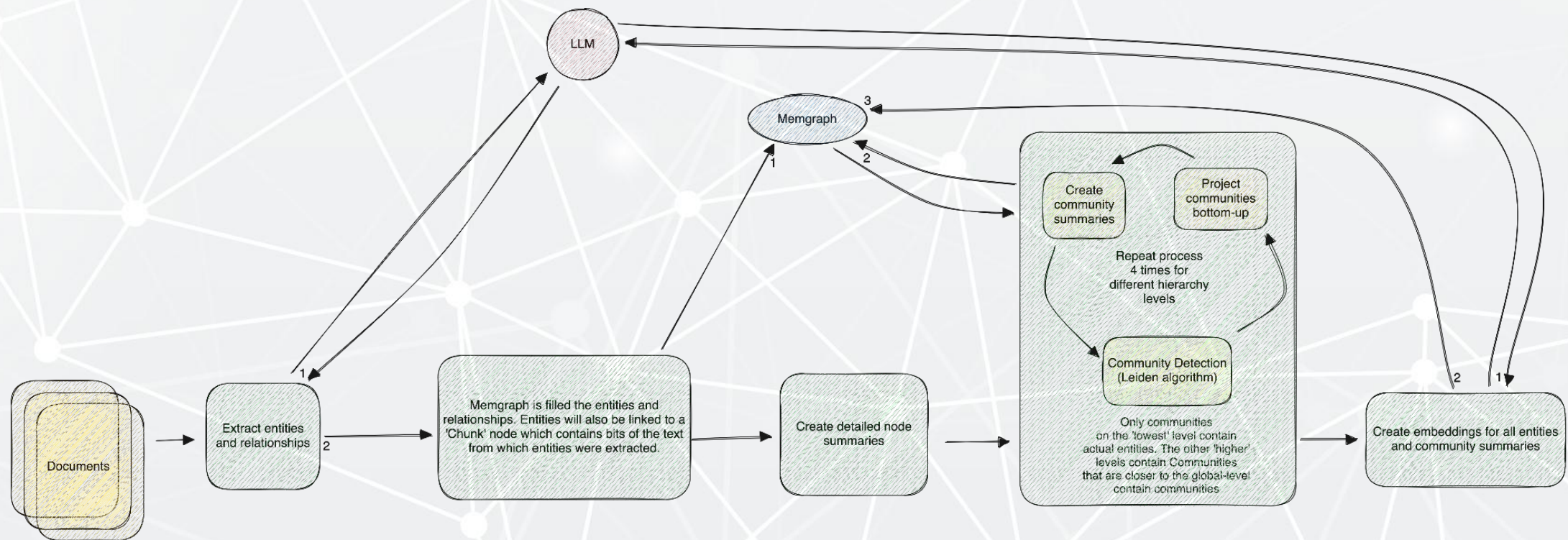
Our Approach

Replicating the Microsoft GraphRAG Approach

- Built on top of LlamaIndex + Memgraph integration: Easily generate the graph from text
 - Extracting entities and relationships from text
 - Upsert data to Memgraph with basic entity resolution
- Utilising Memgraph 3.0's Leiden community detection
 - Group related entities and concepts
 - Build a hierarchical model of information in the graph
- Create community-summaries
 - A short summary for community is created (by summarising over the linked entities)
 - Allows generalisation over large portions of the graph
- During question-answering:
 - Pre-summarised information inside the communities
 - More well-connected graph, organised by concepts

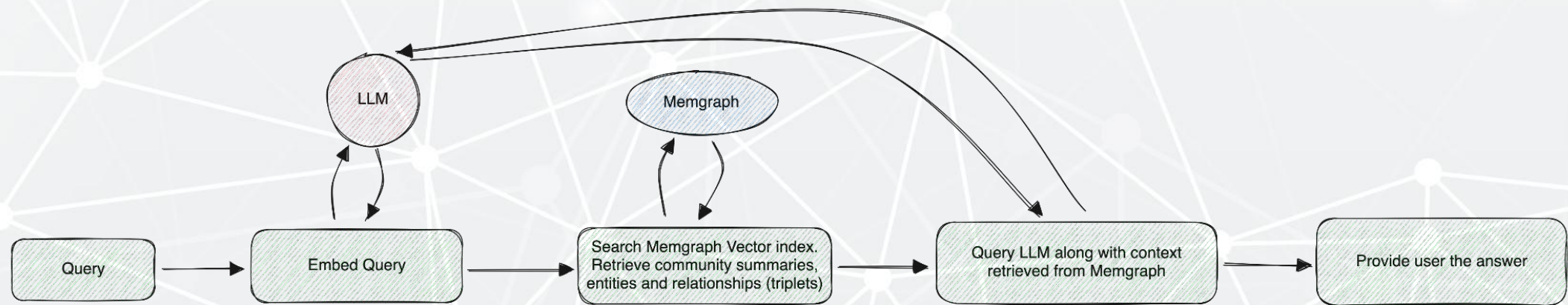
Improved GraphRAG: Creating The Graph

A high-level summary of creating the graph: “The Indexing Process”



Improved GraphRAG: Querying The Graph

How to ask a question over the graph



Step 1: Build The Graph

Use LlamaIndex+Memgraph Integration

- We can easily use some prebuilt tooling for this first step
- Load in the document (to index)
- Set up our LLM (We used the OpenAI API)
 - Completions: GPT-4o-mini
 - Embeddings: text-embedding-3-small (1536 dimensions)
- LlamaIndex tools:
 - SchemaLLMPPathExtractor to fetch the triplet of subject, relation and object
 - MemgraphPropertyGraphStore to connect with our Memgraph database
- Pydantic schema: We provide a schema to ensure that the labels for nodes are according to our specification

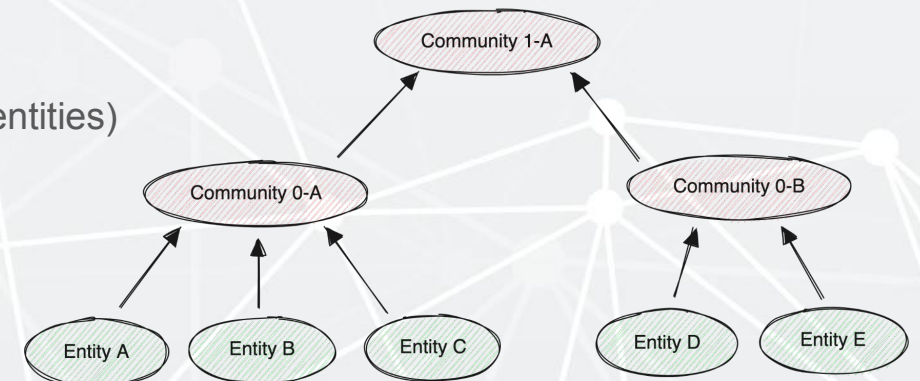
```
graph_store = MemgraphPropertyGraphStore(  
    username=MEMGRAPH_USER,  
    password=MEMGRAPH_PASSWORD,  
    url=MEMGRAPH_URL,  
)  
  
documents = SimpleDirectoryReader("../dataset/the-time-machine").load_data()  
index = PropertyGraphIndex.from_documents(  
    documents,  
    kg_extractors=[  
        SchemaLLMPPathExtractor(  
            llm=OpenAI(model=openai_completions_model, temperature=0.0),  
            max_triplets_per_chunk=20,  
            kg_validation_schema = schema,  
            strict=False  
        )  
    ],  
    property_graph_store=graph_store,  
    show_progress=True,  
)
```

Step 2: Community Detection

Creating the conceptual links between groups of entities

- Utilise the new Leiden community detection algorithm available in Memgraph 3.0
- Creating hierarchical communities:
 - V.A. Traag et al. “From Louvain to Leiden: guaranteeing well-connected communities”. Leiden University. Retrieved from <https://arxiv.org/pdf/1810.08473>
 - Leiden community detection groups well-connected communities together (imagine, similar to clustering)
 - Provides a hierarchy of communities
- Each community is represented by a node, and linked to the communities (or entities) above and beneath it
- The communities are created as ‘community IDs’; a property on each node
 - A node is created for each unique community
 - Actual relationships between the entities and communities in the graph are manually created

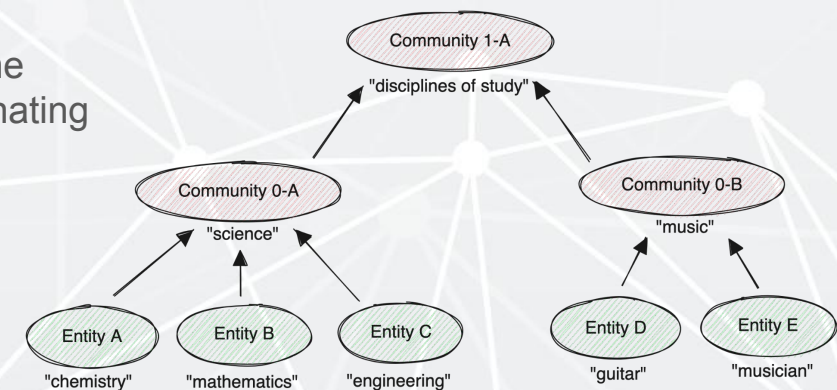
```
// Project our graph,  
// ignoring 'Chunk' (which represents the original texts)  
MATCH p=(m)-[r]->(n)  
WHERE (NOT m:Chunk) AND (NOT n:Chunk)  
WITH project(p) AS subgraph  
// Run community detection  
CALL leiden_community_detection.get(subgraph, "null", 1.0)  
YIELD node, community_id, communities  
SET node.communities = communities
```



Step 3: Create Summaries

Summarise over entire communities

- Start from the 'bottom-up': Start with the entities.
- We create a short description for each entity (considering the original wordsense in context), append it as a 'summary' property on the __Entity__ labelled nodes.
- We then aggregate all the entities under each __Community__ node at the lowest level (community level 0), and create a 'summary' for that community.
- This is done by concatenating the 'summary' texts of all the entities linked to the community and using the LLM to generate an overall community summary. We then attach the newly created summary to the 'summary' property of the community.
- We go one step further. We 'move up' a level to hierarchy level 1, and summarise all the communities beneath it in a similar way by aggregating the sub-communities, concatenating the text and generating a summary of the text using an LLM.
- We continue moving up the hierarchy until we reach the 'top'.
- The number of hierarchical levels depends on:
 - The size of the graph
 - The gamma parameter (configurable) which determines the size of the communities



Step 4: Embeddings

We create the vector indexes in Memgraph and generate embeddings

- In order to be able to search for information later on, we need to create embeddings for the entities and communities
- We create the vector index in Memgraph:

```
CREATE VECTOR INDEX community_index ON :__Community__(embedding) WITH CONFIG {"dimension": 1536, "capacity": 20000};  
CREATE VECTOR INDEX entity_index ON :__Entity__(embedding) WITH CONFIG {"dimension": 1536, "capacity": 20000};
```

- Create embeddings for all entities based on the entity and community summaries
 - Fetch the text using cypher
 - Generate embeddings
 - Upload the embeddings back to the node
- Now, we can query the graph by using vector search! We can search over:
 - Entities: Very localised information
 - Communities: Summarises different portions of the graph

Step 5: Asking The Graph Questions

Build a chatbot that is aware of global-level information in the graph

- Build an interface around this global graph-search
- We can retrieve information about the entire graph as well as localised information
- The key parts of the process are:
 - A paraphrased version of the user query is turned into an embedding (vector) using the LLM
 - Perform vector search in Memgraph with the embedding
 - Retrieve relevant entities and community summaries
 - Allow the LLM to answer the question with the enhanced context

Knowledge Graph Search

The Time Machine by H.G. Wells

☒ Include communities



What are some overarching themes in this book? Keep it very condensed.



Overarching themes in the book include time travel, human evolution, societal decay, the dichotomy between civilization and primal instincts, and the emotional landscape of fear and survival in a dystopian future. Additionally, it explores class disparities and the consequences of technological advancement, reflecting on the complexities of human existence amidst chaos and decay.

What would you like to know?



Other Considerations and Limitations

Possible considerations or future improvements

- Entity-Resolution is limited: Entities are resolved only by name for now (entities with the same name are merged to be the same node)
- Tweaking the gamma parameter during community detection is done experimentally, could be done in a better way
- Indexing is relatively slow (and potentially costly) compared to standard RAG due to entity extraction, and generating summaries. However, inference cost over the long term could be cheaper since contexts fetched from the graph should have higher relevance and informational content compared to standard RAG.
- More advanced graph-traversing algorithms could be incorporated into the solution.
- More advanced methods of querying and ranking (as in the Microsoft implementation) responded were not implemented.



Q&A

redfield.ai





Thank you!

redfield.ai



