PATH-RAG:

reasoning over knowledge paths

By Koorosh Asil

Introduction:

- RAG empowers LLMs to access domain-specific knowledge from external databases, enhancing the response quality without additional training.
- Most RAG approaches divide the text database into chunks, organizing them in a flat structure to facilitate efficient and precise searches
- To better capture the inherent dependencies and structured relationships across texts in a database, researchers use graph-based RAG, which organizes textual information into an indexing graph.
- In this graph, nodes represent entities extracted from the text, while edges denote the relationships between these entities.

graph-rag VS traditional rag ?

- Traditional RAG usually focuses on questions that can be answered with local information about a single entity or relationship.
- In contrast, graph-based RAG targets on global-level questions that need the information across a database to generate a summary-like response.
- For example, GraphRAG first applies community detection on the graph, and then gradually summarizes the information in each community.
- However, we argue that the information considered in previous graph-based RAG methods is often redundant, which can introduce noise, degrade model performance, and increase token consumption.

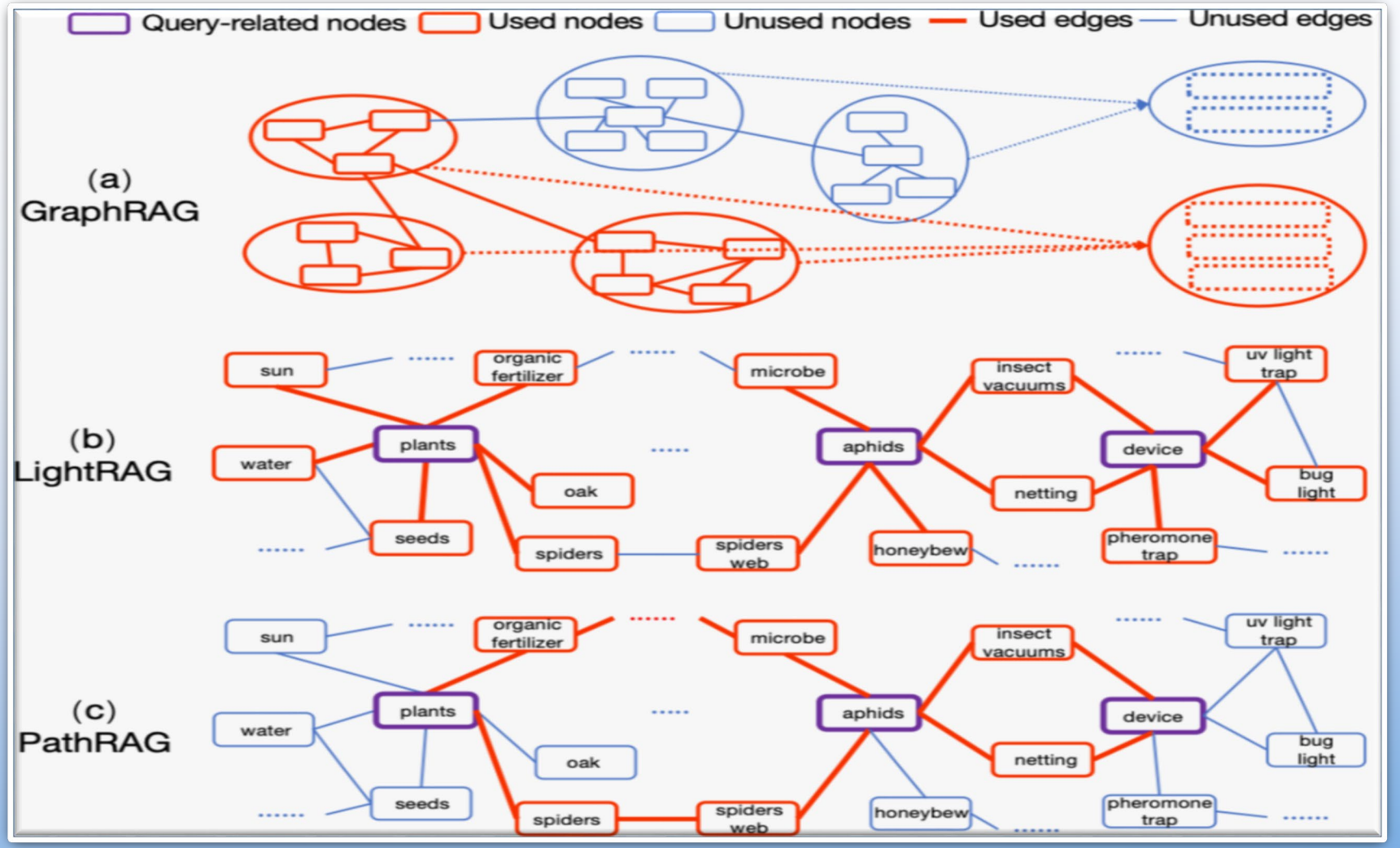


FIGURE 1: COMPARISON BETWEEN DIFFERENT GRAPH-BASED RAG METHODS.

Solution: PATH-RAG

- To overcome the above limitations, we propose PathRAG, which performs key path retrieval among retrieved nodes and converts these paths into textual form for LLM prompting.
- we focus on the key relational paths between retrieved nodes to alleviate noise and re- duce token consumption.
- Specifically, we first retrieve relevant nodes from the indexing graph based on the keywords in the query. Then we design a flow-based pruning algorithm with distance awareness to identify the key relational paths between each pair of retrieved nodes.

“LOST IN THE MIDDLE” problem

- The **lost in the middle** problem is when language models focus more on the **start and end** of long inputs and **ignore important information in the middle**.
- As context length increases, mid-document facts are less likely to affect the output. This hurts long-document QA and RAG, motivating chunking and structured retrieval.
- HOW TO SOLVE IN PATH-RAG?

  we sequentially concatenate the node and edge information alongside each path as textual relational paths. we place the textual paths into the prompt in ascending order of reliability. scores for better answer generation.

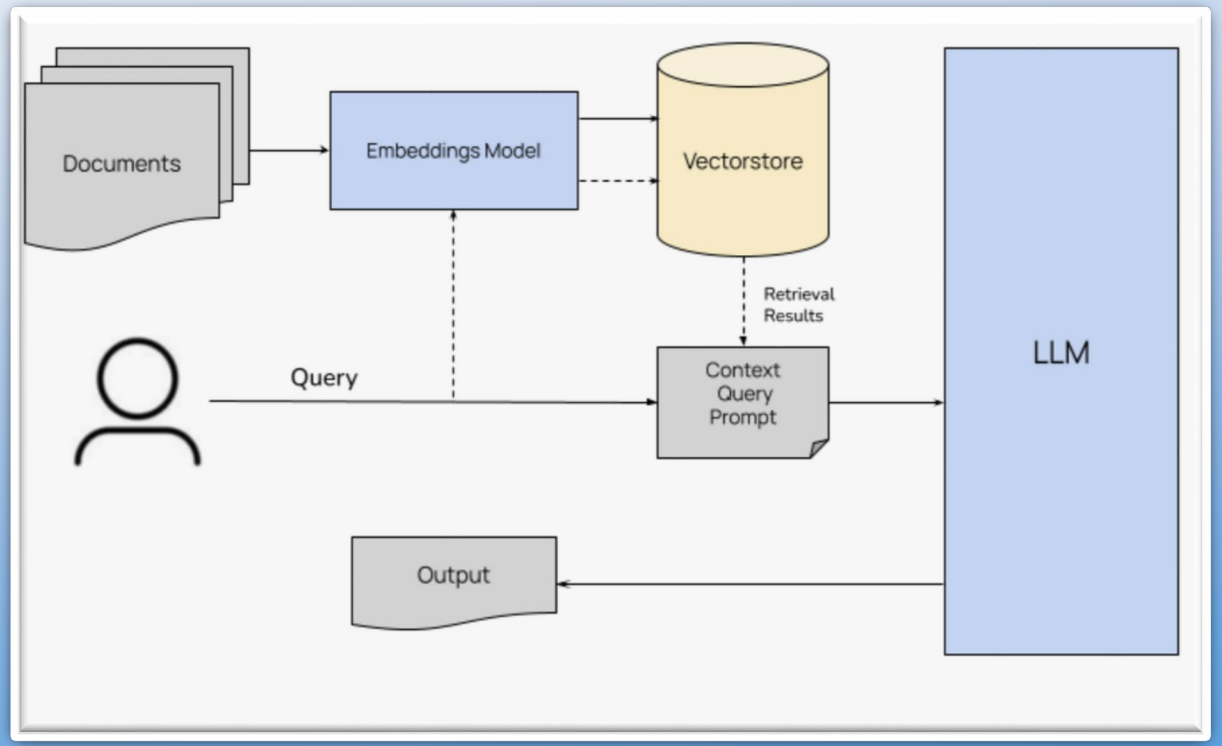
KNOWLEDGE SOURCES FOR RAG

- A **knowledge base** is an external repository that stores structured or unstructured information used by retrieval-augmented generation systems to support language models.
- Instead of relying solely on parametric memory, LLMs query these knowledge sources to obtain relevant facts, context, or relationships at inference time.
- Knowledge bases can take different forms—such as text collections, knowledge graphs, or graph datasets—each offering distinct ways of organizing and accessing information for downstream reasoning and generation tasks.

Text-Based RAG

- Text-based RAG is widely used in LLMs to improve text quality and reduce hallucinations by retrieving information from external textual databases that store large amounts of domain knowledge.

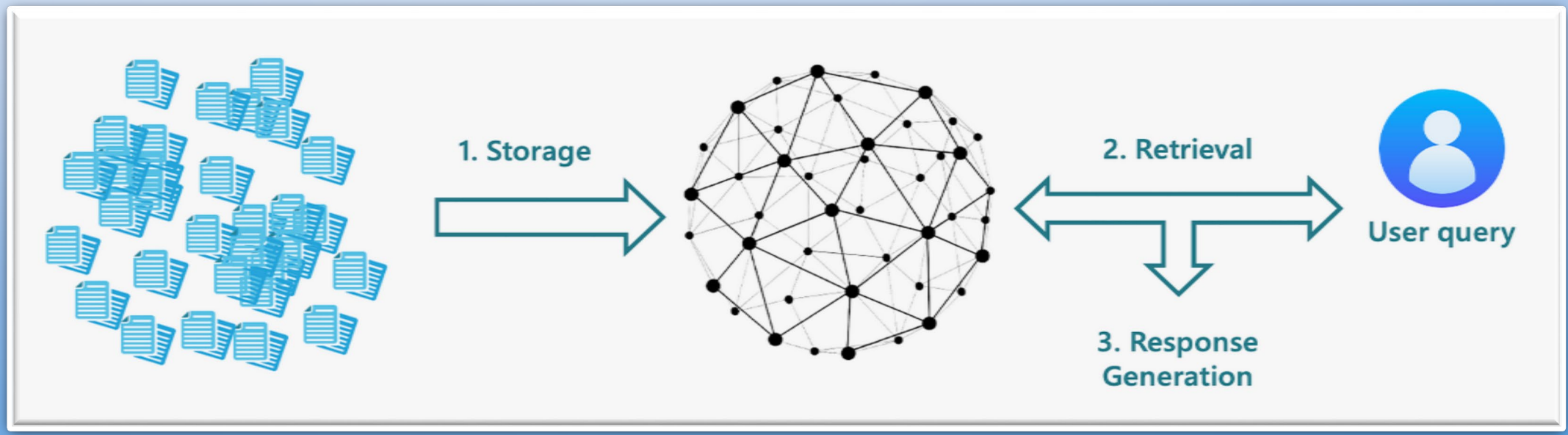
- Text-based RAG systems rely on **textual data only**, allowing LLMs to directly retrieve and condition on retrieved text chunks.



- Based on retrieval mechanisms, **text-based RAG is divided into two categories**:
  I. **Sparse vector retrieval**: identifies representative words using word frequency and retrieves text via **keyword matching**.
  II. **Dense vector retrieval**: encodes queries and text into **vector embeddings** and retrieves content based on **embedding similarity**, addressing lexical mismatch and synonym issues.
- Most existing text-based RAG methods use a **flat organization of text chunks** and **do not model relationships or contextual dependencies between chunks**, which limits the quality of LLM-generated responses.

KG-RAG

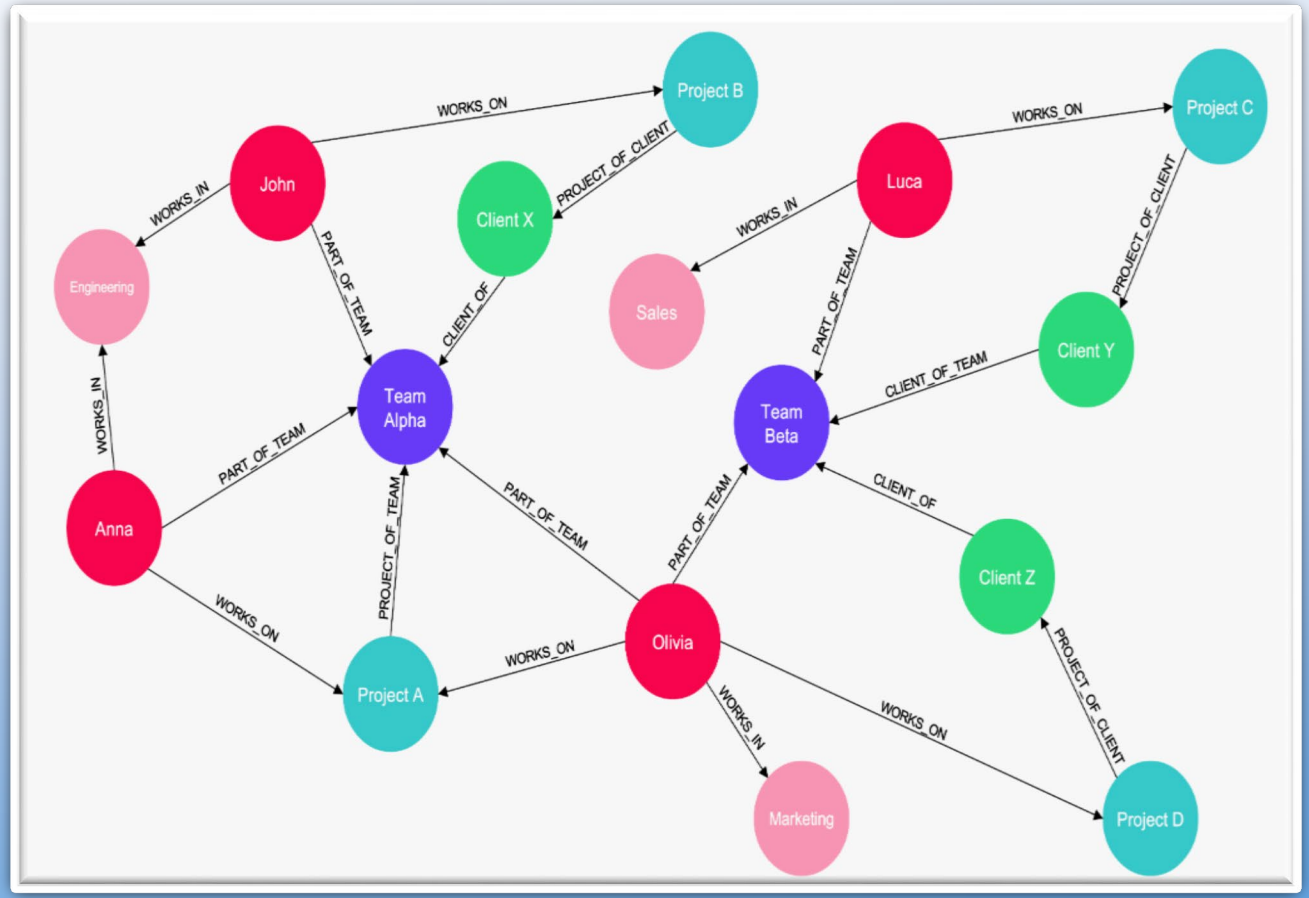
- **KG-RAG** retrieves information from **knowledge graphs (KGs)** instead of text databases.
- It can use **existing KGs** or **optimized/enhanced versions** of KGs.
- KG-RAG enables LLMs to retrieve **entities and their relationships**, providing structured knowledge for generation



- KG-RAG methods typically **extract a local subgraph** from the KG, such as **immediate neighbors of queried entities**.

- This approach focuses on **entity-centric retrieval**.

- Most KG-RAG methods handle questions involving **a single entity or relation**, which **limits their applicability** to more complex, multi-hop reasoning tasks.



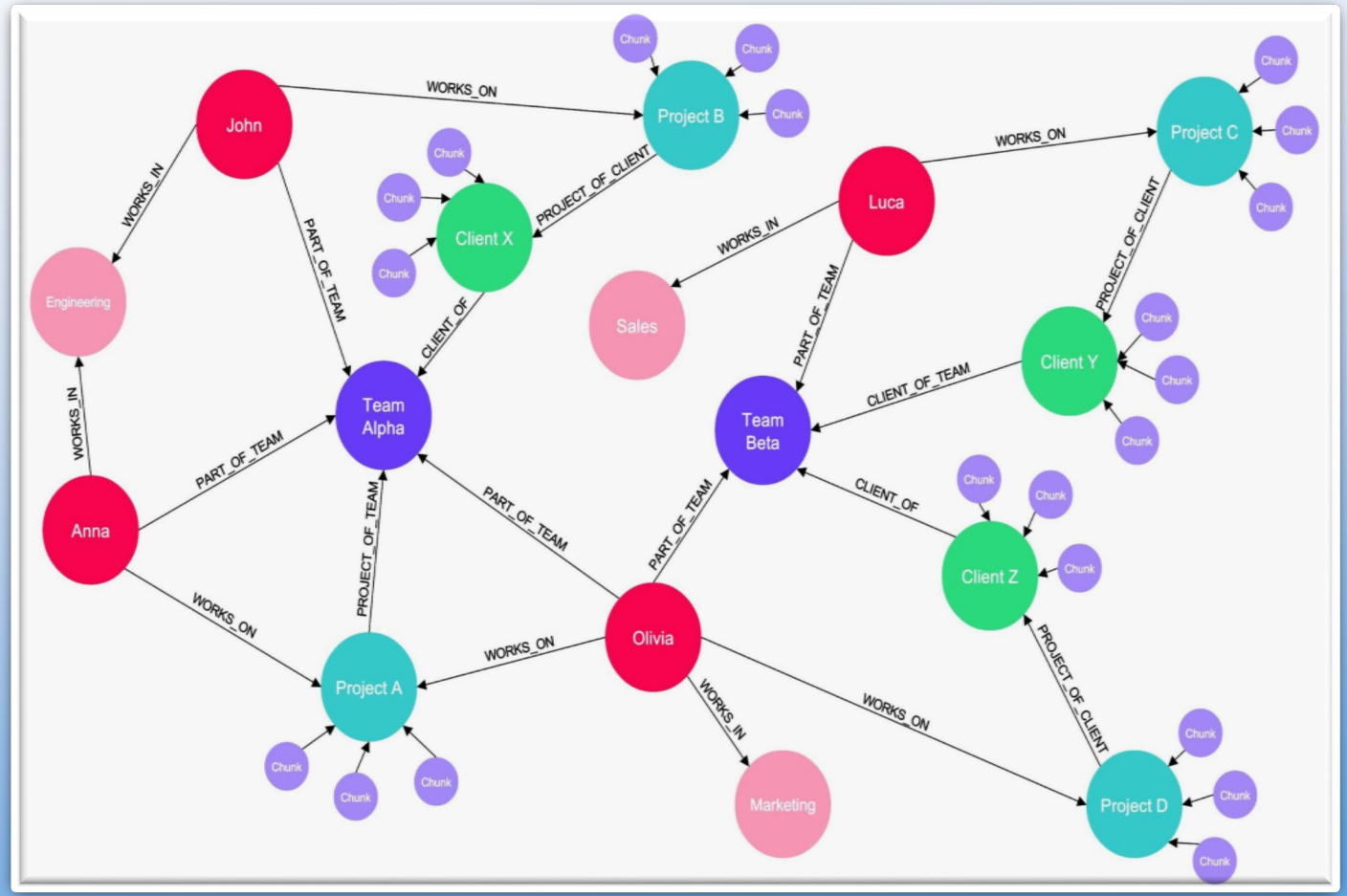
Graph-based RAG

- **Graph-based RAG** organizes text databases as **text-associated graphs** instead of using preconstructed knowledge graphs.
- It targets **global-level questions** that require information from **multiple text segments** across a database.
- Graph construction involves **extracting entities** from text and **identifying relationships** between them.
- **Contextual text** is preserved as descriptions to reduce information loss during text-to-graph conversion.

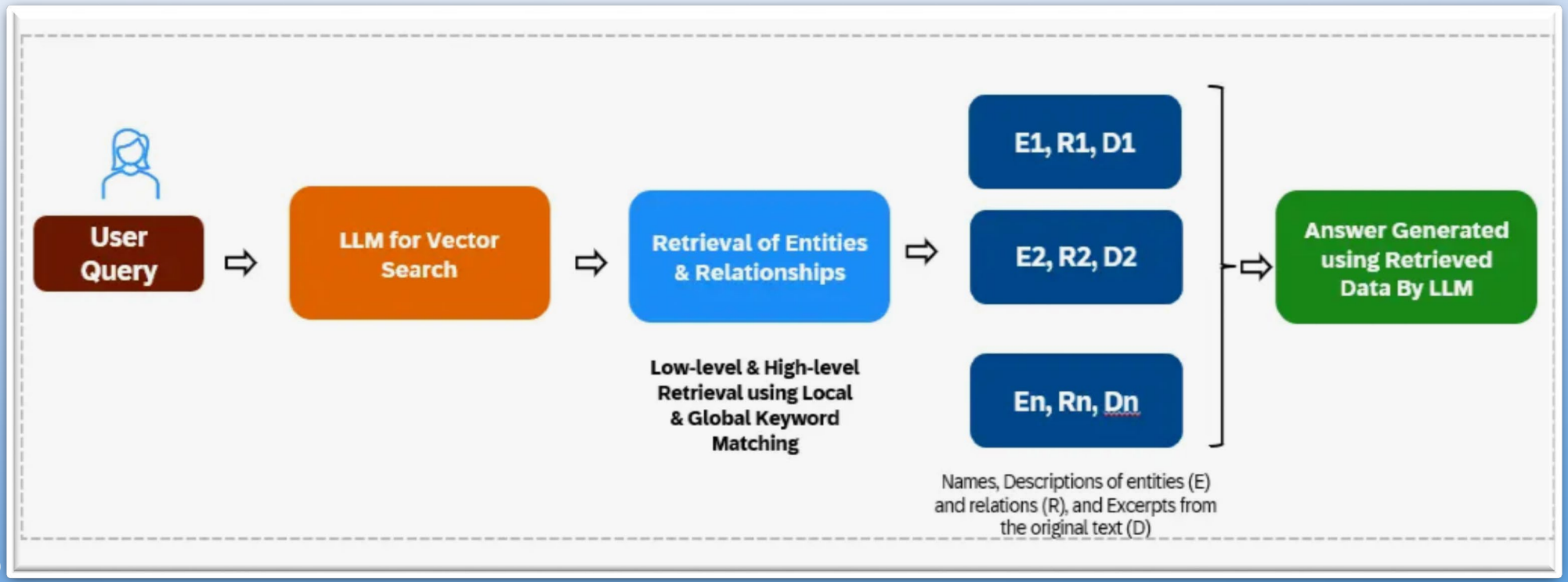
- **GraphRAG** applies **community detection algorithms** on the constructed graph.

- Information is **aggregated hierarchically**, from **sub-communities** to higher-level communities.

- This enables **multi-segment and global information integration** for downstream generation.



- **LightRAG** adopts a **dual-stage retrieval framework** to accelerate the retrieval process.
- It extracts both **local and global keywords** from the input question.
- Relevant **nodes and edges** are retrieved using these keywords.



- The **ego-network** of retrieved nodes is treated as the **final retrieval result**.
- This approach **simplifies retrieval** and effectively supports **global-level tasks**.
- However, retrieving **all immediate neighbors** may introduce **noise**, which can negatively affect answer quality

Indexing-Graph construction

- **LLMs identify entities and interrelations** within the text database
- The indexing graph is defined as $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{K_v}, \boldsymbol{T})$
  - **V (nodes):** entities
  - **E (edges):** relationships between entities
- Each node **v ∈ V** includes:
  - **Identifier** $\boldsymbol{K_v}$ (e.g., entity name)
  - **Textual chunk** $\boldsymbol{t_v}$ (associated text)
- Each edge $\boldsymbol{e} \in \boldsymbol{E}$ includes:
  - A **descriptive textual chunk** $\boldsymbol{t_e}$ to enrich relational context

- Given a query **q**, a **graph-oriented retriever** selects relevant **nodes and edges** from the indexing graph.
- The **textual chunks** of retrieved nodes and edges are combined with the query.
- An **LLM generator** produces the final answer using the retrieved graph context.
- The graph-based RAG process is summarized as:
  - $A(q, G) = F \circ M(q; R(q, G)$
  - Where:
    - **R(q, G):** graph-oriented retriever
    - **M:** prompt template
    - **F:** LLM generator
    - **A:** retrieval-augmented generation output

Node Retrieval

- Node retrieval identifies **relevant graph nodes** based on the input query.
- An **LLM extracts keywords** from the query.
- The extracted keywords are denoted as $\boldsymbol{K_q}$.
- **Dense vector matching** is used to retrieve nodes from the indexing graph **G**.
- Relevance between a keyword and a node is measured by **semantic similarity**.
- **Cosine similarity** is used in the embedding space.

- A semantic embedding model f encodes:
  - Query keywords: $K_q \rightarrow X_q$
  - Node identifiers: $K_V \rightarrow X_V$
- $X_V = \{x_v \mid v \in V\}$:= embeddings of node identifiers
- $X_q = \{x_{q,i}\}, i = 1 \ldots |X_q|$: embeddings of extracted keywords
- For each keyword embedding in X_q, the most relevant nodes are retrieved from X_v.
- Retrieval continues until a predefined number **N** of nodes is reached.
- The final retrieved node set is denoted as $V_q \subseteq V$.

Node Retrieval

- Multiple paths may exist between retrieved nodes.
- Not all paths are useful for answering the query.
- A **flow-based pruning algorithm** is used to extract **key paths** efficiently.
- The method incorporates **distance awareness** to favor closer and stronger connections.

Definitions

- For a node v_i:
  - $N(v_i, \cdot) :=$ nodes that v_i **points to**
  - $N(\cdot, v_i) :=$ nodes that **point to** v_i
- Each node is assigned a **resource score** $S(v_i)$.
- Initialization:
  - $S(v_{start}) = 1$
  - All other nodes: $S = 0$

Resource propagation rule

- Resources are propagated through the graph using:

$$S(v_i) = \sum_{v_j \in N(\cdot, v_i)} \frac{\alpha\, S(v_j)}{|N(v_i, \cdot)|}$$

- α is a decay factor controlling information loss over distance.
- Resources are split among outgoing neighbors.
- Nodes closer to the start node receive more resource, while distant nodes receive less.

- The decay factor **penalizes distant nodes**, making the retriever sensitive to graph distance.
- Stronger, shorter connections accumulate more resource.
- Paths with higher total resource are considered **more important**.

- This method does **not rely on fixed hop limits**.
- Path importance is determined by **connection strength**, not just path length.
- Enables flexible and effective pruning for complex graph structures.

Stopping criterion

- An **early stopping strategy** is introduced to prune paths during propagation.
- A node $\mathbf{v_i}$ is pruned when:

$$\frac{S(v_i)}{|N(v_i)|} < \theta$$

- θ is a predefined **pruning threshold**.
- The resource is normalized by the node's out-degree to measure its actual contribution.

Path reliability measurement

- For each path **P**, a **reliability score** is computed.
- Reliability is defined as the **average resource value** along the path:

$$S(P) = \frac{1}{|\mathcal{E}_P|} \sum_{v_i \epsilon P} S(v_i)$$

- Each path consists of:
- V_P: set of nodes in the path
- $\mathcal{E}_P$: set of directed edges in the path
- Paths passing through higher-resource nodes are considered more reliable.

Textual relational path construction

- Each retrieved **relational path** is converted into a **textual form**.
- This is done by **concatenating textual chunks** from:
  - Nodes along the path
  - Edges connecting the nodes
- A textual relational path is defined as:

$$t_P = concat(\dots; t_{v_i}; t_{e_i}; t_{v_{i+1}}; \dots)$$

- Where:
  - t_{v_i} := text associated with node v_i
  - $\boldsymbol{t_{e_i}}$:= text describing edge e_i

- LLMs tend to focus more on information at the **beginning and end** of long prompts.
- These positions are treated as the **“golden memory region”** for model comprehension.
- Prompt structure is explicitly designed to exploit this behavior.
- The **query** is placed at the **beginning** of the prompt.
- Retrieved **textual relational paths** are sorted by **reliability**.
- The final prompt is defined as:

$$M(q; R(q, G)) := concat([q; t_{P_k}; \ldots; t_{P_1}])$$

complexity analysis – node propagation

- Due to **decay penalty** and **early stopping**, the number of active nodes after step i is at most:

$$\frac{\alpha^i}{\theta}$$

- Total nodes involved in propagation:

$$\sum_{i=0}^{\infty} \frac{\alpha^i}{\theta} = \frac{1}{(1-\alpha)\theta}$$

- Extracting candidate paths between all node pairs has complexity:

$$O\left(\frac{N^2}{(1-\alpha)\theta}\right)$$

Evaluation

- Due to the absence of ground truth answers, we follow the LLM-based evaluation procedures. we utilize“GPT-4o-mini” to evaluate the generated answers across multiple dimensions.
- We compare the answers generated by each baseline and our method and conduct win-rate statistics. A higher win rate indicates a greater performance advantage over the other.
- PathRAG consistently outperforms the baselines across all evaluation dimensions and datasets.

PathRag win Rate across all metrics

Comprehensiveness	Diversity	Logicality	Relevance	Coherence
60.88%	62.75%	59.78%	60.47%	59.93%

- Comprehensiveness: how fully the answer covers all important aspects of the question, without missing key points.
- Diversity: how well the answer incorporates different perspectives, facts, or reasoning paths instead of repeating the same idea.
- Logicality: how logically sound and well-reasoned the answer is, with clear cause-and-effect or step-by-step reasoning.
- Relevance: how directly the answer addresses the question and avoids unnecessary or off-topic information.
- Coherence: how clear, fluent, and well-structured the answer is as a whole, with ideas flowing naturally.

hyperparameter analysis: (legal dataset)

- Number of retrieved nodes (N):
  Performance improves as N increases, peaks at N = 40, then slightly declines because extra nodes become less relevant and introduce noise.
- Number of retrieved paths (K):
  Performance peaks at K = 15; adding more paths (e.g., K = 25) does not help and can hurt performance, though larger datasets may benefit from larger K.
- Decay rate (α):
  Best performance occurs at α = 0.8; too small α over-prioritizes short paths, while α = 1.0 ignores distance and significantly degrades performance.

Thanks for listening.