# GraphRAG

"From Local to Global: A Graph RAG Approach to Query-Focused Summarization."

chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://arxiv.org/pdf/2404.16130
https://medium.com/@zilliz_learn/graphrag-explained-enhancing-rag-with-knowledge-graphs-3312065f99e1

## Issues with RAG

- While RAG based approach works well in many cases, it struggles with complex tasks like
- multi-hop reasoning (Needs to **combine facts** across different entities) or
- answering questions that require connecting disparate pieces of information.

Consider this question: “***What name was given to the son of the man who defeated the usurper Allectus?***”

A baseline RAG would generally follow these steps to answer this question:

1. Identify the Man: Determine who defeated Allectus.
2. Research the Man’s Son: Look up information about this person’s family, specifically his son.
3. Find the Name: Identify the name of the son

- The challenge usually arises at the first step because a baseline RAG retrieves text based on semantic similarity, not directly answering complex queries where specific details may not be explicitly mentioned in the dataset.
- This limitation makes it difficult to find the exact information needed, often requiring expensive and impractical solutions like manually creating Q&A pairs for frequent queries.

Let's say we retrieve these:

- **Chunk 1:** "Allectus was defeated by Constantius Chlorus in 296 AD."
- **Chunk 2:** "Constantine the Great was the son of Constantius Chlorus."
- **Chunk 3:** "Constantius Chlorus was a Roman emperor."

Now here's the trick: **none of these chunks were written to be stitched together**. They're independent, raw facts.

> ✅ **A human could say:**
> **"Oh! So if Constantius defeated Allectus, and Constantine is his son... then Constantine is the answer."**

But an LLM is not always good at:

- **Tracking coreference:** "the man who defeated Allectus" = Constantius Chlorus
- **Following reasoning chains:** "find that man → now find his son → now get the son's name"
- **Handling indirect, multi-step logic** when it's spread across multiple documents

Baseline RAG treats the top-k as **flat context**.
It has no clue which fact **connects to what**, or in what **order** to reason.
There's **no structure, no link traversal, no guided reasoning**.
Even if the LLM can *sometimes* pull it off, it's:

- **Unreliable**
- Often **hallucinates**
- Depends on **chance ordering** and **token limits of the response.**

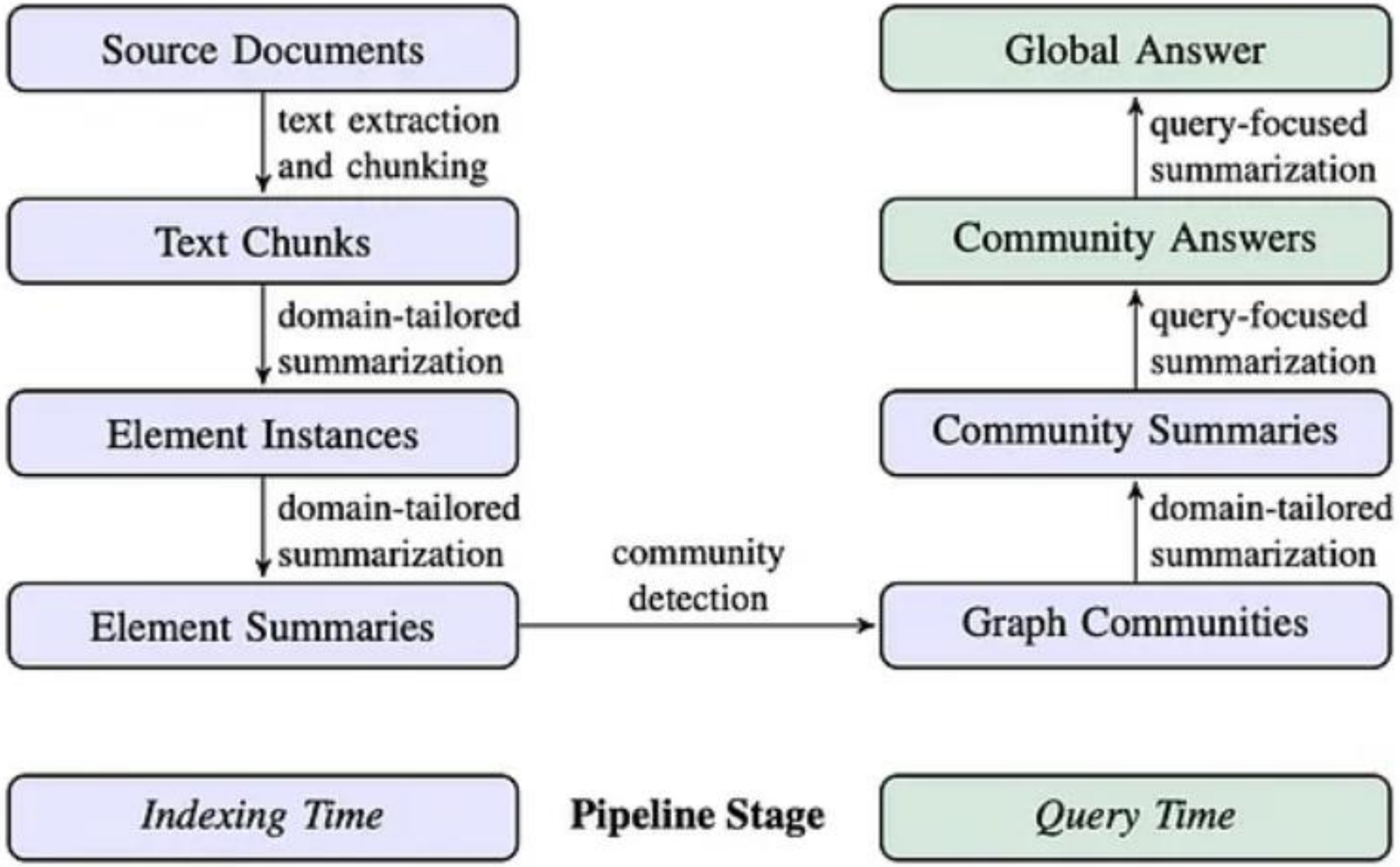
| Failure Type | Description |
|---|---|
| **Disconnected chunks** | Facts exist but are in separate, unlinked chunks |
| **Coreference ambiguity** | LLM can't link "the man" → Constantius without help |
| **Missing a hop** | One key fact is missing, breaking the reasoning chain |
| **Chunk prioritization error** | Irrelevant but similar text is retrieved instead of key fact |
| **No reasoning structure** | LLM is not told how to combine facts |
| **Token limit issues** | Retrieved info is cut off before all needed facts are used |

## GraphRAG

- GraphRAG is a method that augments RAG retrieval and generation with knowledge graphs.
- Unlike a baseline RAG that uses a vector database to retrieve semantically similar text, GraphRAG enhances RAG by incorporating knowledge graphs (KGs).
- Knowledge graphs are data structures that store and link related or unrelated data based on their relationships.

## Knowledge Graphs

- **What it is**:
  - A structured, graph-based representation of **entities (nodes)** and their **relationships (edges)**.
  - Built using **triplets** like `(Entity A, relationship, Entity B)` (e.g., `(Tesla, founded_by, Elon Musk)`).
- **Key Features**:
  - Stores **explicit, granular data** (e.g., entity attributes, hierarchies, connections).
  - Supports **complex queries** (e.g., *"Find all companies founded by Elon Musk"*).
  - Enables **graph traversal** (e.g., navigating from *Tesla* to *Elon Musk* to *SpaceX*).
- **Role in GraphRAG**:
  - Acts as the **source of truth for entity-level details**.
  - Used during **local search** to resolve specific entities and their relationships after a community is retrieved.


Source Documents
text extraction and chunking
Text Chunks
domain-tailored summarization
Element Instances
domain-tailored summarization
Element Summaries
community detection
Graph Communities
domain-tailored summarization
Community Summaries
query-focused summarization
Community Answers
query-focused summarization
Global Answer
Indexing Time
Pipeline Stage
Query Time

**Indexing**: This is your **preprocessing phase**.

## 1.Source Documents → Text Chunks

1. Basic chunking (paragraphs, sentences, sections).
2. Think of this as raw document segmentation.

## 2.Text Chunks → Element Instances

1. Each chunk is passed through an LLM to **extract structured data**:
    1. **Entities** (people, orgs, places)
    2. **Relationships** (e.g., “X works at Y”, “Z founded A”)
    3. **Key claims** (statements/assertions — e.g., “Company X raised $10M in 2023”)

**Key claims** are like atomic facts or opinions from the text that can stand on their own, often expressed as subject-predicate-object triplets.

## 3. Element Instances → Element Summaries

1. These are **mini summaries** of the info from each chunk (entities, rels, claims).
2. This step helps “boil down” each unit into graph-compatible form.

## 4. Element Summaries → Graph Communities

1. Graph is built with nodes (entities/claims) and edges (relationships).
2. Then **community detection (e.g., Leiden algorithm)** is run:
    1. Grouping entities that are densely connected = **communities**.
    2. E.g., a community could be all entities in a startup ecosystem.

# 1. What are key claims/triplets?

You can think of these as **structured semantic units** extracted from unstructured text. Each triplet is:
🟣 **(Subject, Predicate, Object)**
🔶 Also called an **SPO triple** or a **key claim**.
They capture **facts**, **assertions**, or **relationships** in a structured way.
🔷 **Example from text:**
"OpenAI released ChatGPT in November 2022."
Triplet becomes:
→ **(OpenAI, released, ChatGPT)**
→ Or even: **(ChatGPT, was released_in, November 2022)**
These triples are the **building blocks of knowledge graphs**.

They also generalize to claims, like:
•**(Dr. Khan, believes, AI will transform education)** → this is a **subjective claim**
•**(Vitamin D, reduces, risk of infection)** → a scientific claim

## ✅ In GraphRAG:

LLMs extract **many triplets** from each text chunk:

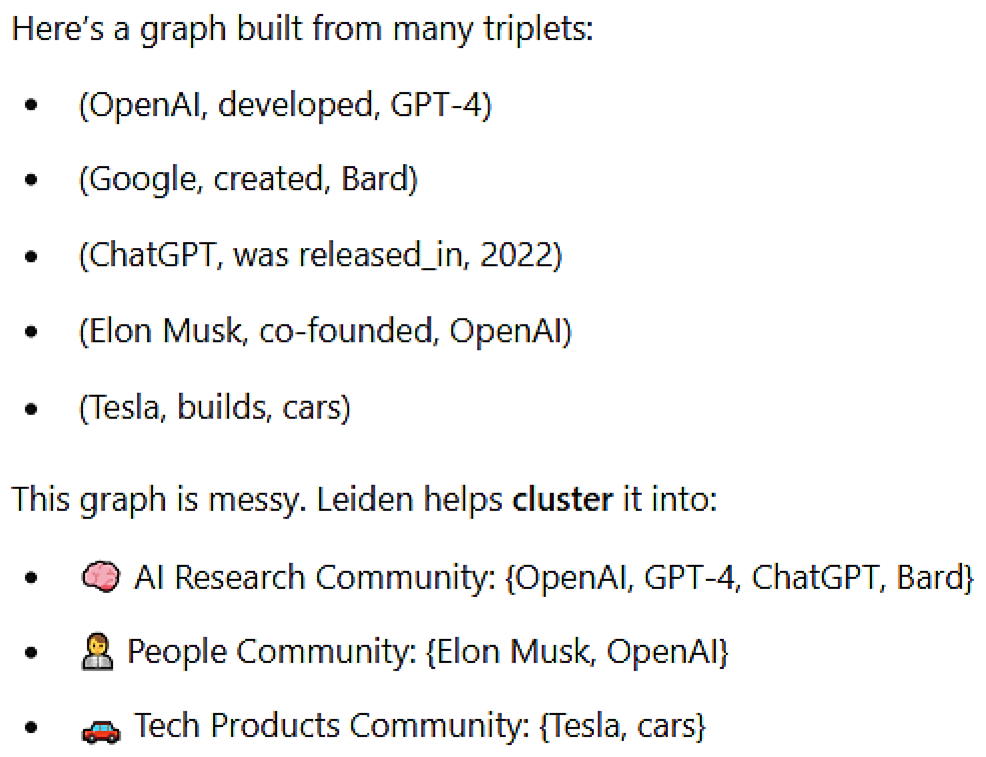
- Entities = Subject/Object
- Predicate = Relationship
- All together = structured **key claims**

# 2. Leiden Algorithm

The Leiden algorithm is used for **community detection** in graphs.

- **What is a *community*?**

**A community** is a group of nodes (entities/claims) that are **densely connected to each other**, but **sparsely connected** to other groups.
This is useful for **breaking up large graphs** into meaningful, thematic sections.



Here's a graph built from many triplets:

- (OpenAI, developed, GPT-4)
- (Google, created, Bard)
- (ChatGPT, was released_in, 2022)
- (Elon Musk, co-founded, OpenAI)
- (Tesla, builds, cars)

This graph is messy. Leiden helps **cluster** it into:

- 🧠 AI Research Community: {OpenAI, GPT-4, ChatGPT, Bard}
- 🧑‍💼 People Community: {Elon Musk, OpenAI}
- 🚗 Tech Products Community: {Tesla, cars}

•For each node, Leiden checks:
*"If I move this node to a neighbor's group, will that create a tighter, more connected community?"*
If yes → it moves the node.
•**Refine Groups**:
After some moves, Leiden checks:
*"Are all my current groups fully connected inside?"*
If any group is **disconnected**, Leiden **splits it** into smaller connected ones.

## 3. 🧩 How do Graph Communities Help?

Once you've got clusters like:

- Community A: {entities + claims about AI models}
- Community B: {people + orgs}
- Community C: {policy + law claims}

You can:

- Generate **summaries per community**
- Use these summaries to answer **broad or deep** questions later (Global Search)
- Or search **within a community** (Local Search)

It's like organizing your textbook **by themes**, and later flipping to the right section.

## Indexing Phase

1. **Text Unit Segmentation**: The entire input corpus is divided into multiple text units (text chunks). These chunks are the smallest analyzable units and can be paragraphs, sentences, or other logical units. By segmenting long documents into smaller chunks, we can extract and preserve more detailed information about this input data.
2. **Entity, Relationship, and Claims Extraction**: GraphRAG uses LLMs to identify and extract all entities (names of people, places, organizations, etc.), relationships between them, and key claims expressed in the text from each text unit. We will use this extracted information to construct an initial knowledge graph.
3. **Hierarchical Clustering**: GraphRAG uses the Leiden technique to perform hierarchical clustering on the initial knowledge graphs. Leiden is a **community detection** algorithm that can effectively discover community structures within the graph. Entities in each cluster are assigned to different communities for more in-depth analysis. If any group is **disconnected**, Leiden **splits it** into smaller connected ones.
4. **Community Summary Generation**: GraphRAG generates summaries for each community and its members using a bottom-up approach. These summaries include the main entities within the community, their relationships, and key claims. This step gives an overview of the entire dataset and provides useful contextual information for subsequent queries.

## How Triples Become a Graph

```
Chunk from article:
"Dr. Smith discovered a new variant of the virus in March 2023."


Extracted:
(Smith, discovered, new variant)
(variant, appeared_in, March 2023)
(variant, is_a, virus)
```

Graph:

- **Nodes**: {Smith, variant, virus, March 2023}
- **Edges**: labeled arrows based on the predicates

Then you cluster the graph into groups based on connectivity — that's where Leiden comes in.

| Step | Diagram Box | What's Happening |
|---|---|---|
| **1. Text Unit Segmentation** | `Source Documents → Text Chunks` | Chunking the corpus |
| **2. Entity, Rel, Claim Extraction** | `Text Chunks → Element Instances` | LLM extracts graph-friendly triples |
| **3. Hierarchical Clustering** | `Element Summaries → Graph Communities` | Clustering via Leiden to find communities |
| **4. Community Summary Generation** | `Graph Communities → Community Summaries` | Bottom-up summarization per community |

## Query Phase

Where the actual **user question gets processed**.

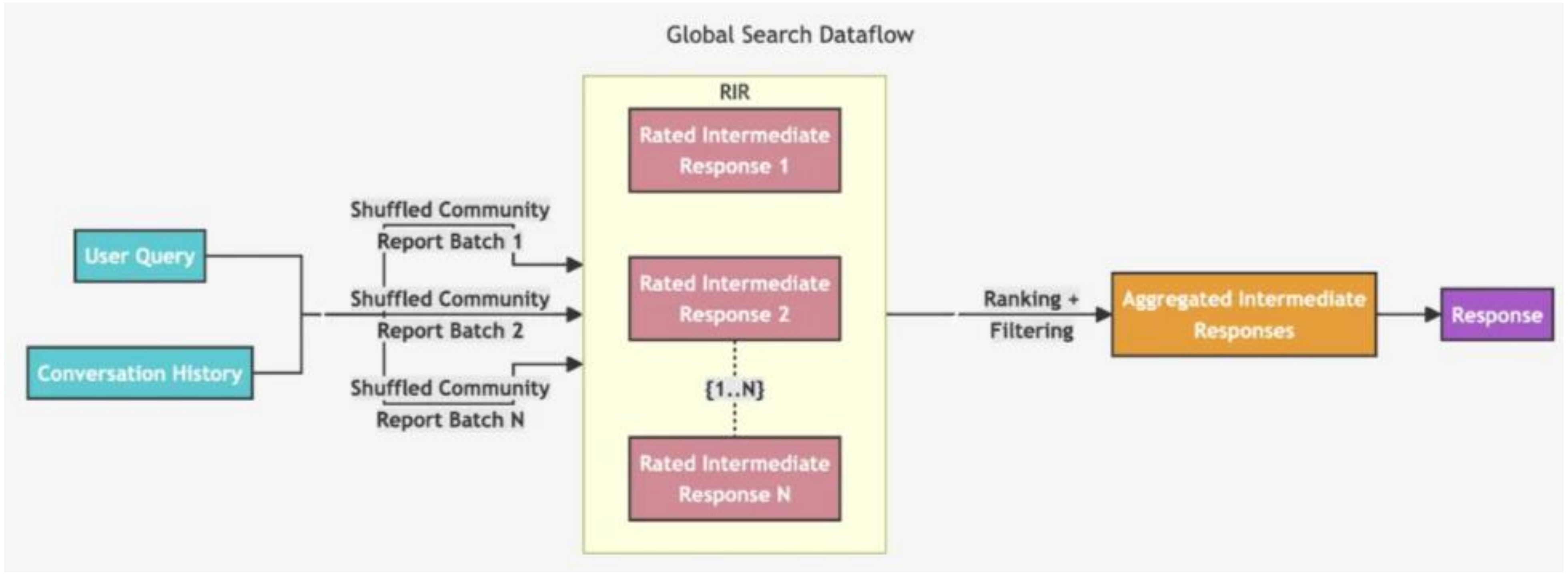
1. **Graph Communities → Community Summaries**
   1. Each community has a **bottom-up summary** (collected info about its entities, rels, claims).
2. **Community Summaries → Community Answers**
   1. For a given query, only the **relevant communities** are picked and used to generate answers.
   2. This is either through **global summarization** or local retrieval.
3. **Community Answers → Global Answer**
   1. The final result, either answering a broad (global) question or detailed (local) one.

## Query Phase

GraphRAG has two different querying workflows tailored for different queries.

- Global Search for reasoning about holistic questions related to the whole data corpus by leveraging the community summaries.
- Local Search for reasoning about specific entities by fanning out to their neighbors and associated concepts.

# Global Search – **For Holistic, Broad Questions**


Global Search Dataflow
User Query
Conversation History
Shuffled Community Report Batch 1
Shuffled Community Report Batch 2
Shuffled Community Report Batch N
RIR
Rated Intermediate Response 1
Rated Intermediate Response 2
{1..N}
Rated Intermediate Response N
Ranking + Filtering
Aggregated Intermediate Responses
Response

# Global Search – For Holistic, Broad Questions

🔷 **Use Case:**

- When the query needs an overview across **multiple topics, communities, or the entire corpus**.
- Example: *"What are the main trends in AI research post-2020?"*

**Steps:**

1. **User Query + History**: User asks a question; chat history helps with context.
2. **Community Reports (Summaries)**: GraphRAG picks summaries of different **graph communities** (clusters of related concepts). The user query (and history) is embedded and matched semantically against these summaries.
3. **Shuffling & Batching**: These community summaries are shuffled to avoid order bias and split into batches  Each batch will be processed separately to improve information density.
4. **Rated Intermediate Responses (RIRs)**:

For **each batch**, GraphRAG:

- Sends the community summaries and the query to an **LLM**.
- The LLM performs two tasks:
  - **Extracts points** (i.e., factual claims, relevant details).
  - **Assigns a score (0–1)** to each point, representing its **importance or relevance** to the query.

**5. Ranking & Aggregation**:

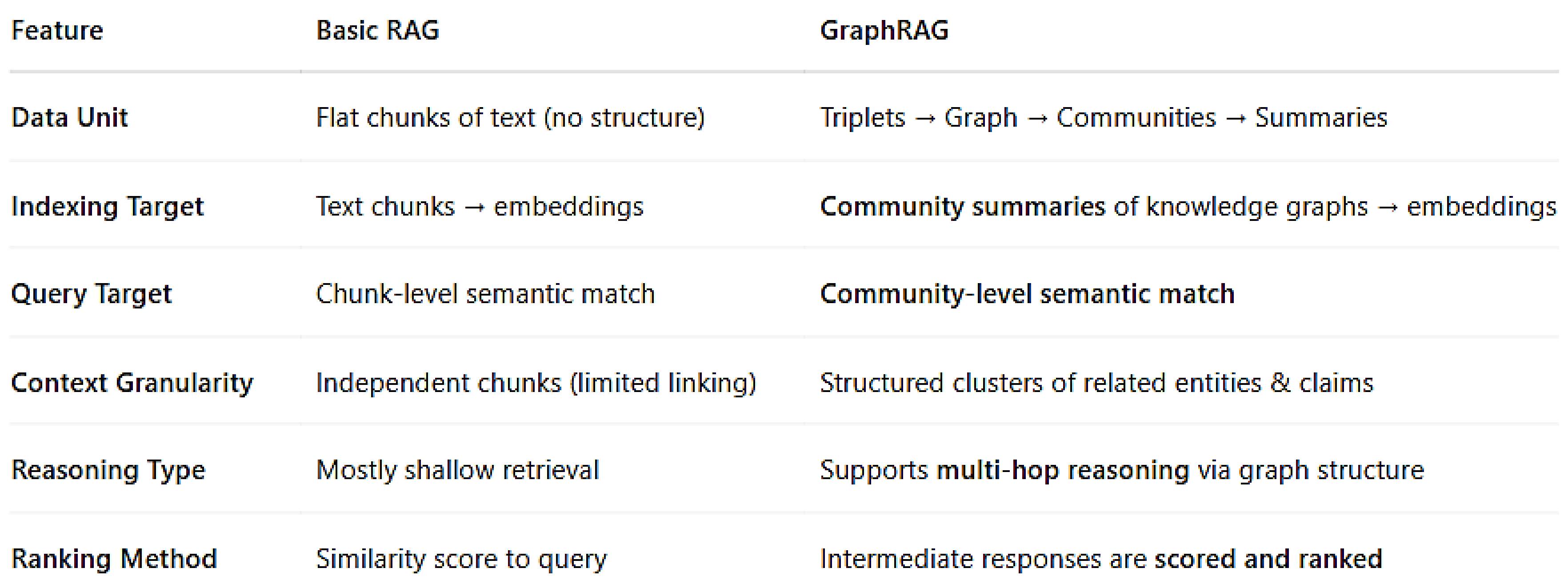
1. Points from all RIRs are ranked.
2. Most important points (top N or filter out below a threshold) are selected into **Aggregated Intermediate Responses**.

**6. Final Answer Generation**:

The **aggregated points** are now used as the **context** for the LLM.

The LLM generates the final answer, guided by:

- the **user query + history**,
- the **most relevant, pre-scored points**,
- ensuring the output is **grounded**, **multi-hop**, and **fact-rich**.



| Feature | Basic RAG | GraphRAG |
|---|---|---|
| Data Unit | Flat chunks of text (no structure) | Triplets → Graph → Communities → Summaries |
| Indexing Target | Text chunks → embeddings | **Community summaries** of knowledge graphs → embeddings |
| Query Target | Chunk-level semantic match | **Community-level semantic match** |
| Context Granularity | Independent chunks (limited linking) | Structured clusters of related entities & claims |
| Reasoning Type | Mostly shallow retrieval | Supports **multi-hop reasoning** via graph structure |
| Ranking Method | Similarity score to query | Intermediate responses are **scored and ranked** |

## What’s Actually Smarter in GraphRAG?

- Instead of comparing your query with hundreds of **independent chunks**, it compares to **community summaries** that are already **structured, distilled**, and **relationally grouped**.
- This makes the **retrieved context much more relevant and coherent**, especially for questions that involve **relationships**, **causes**, **timelines**, or **entity interactions**.

## What Ranks the RIRs (Rated Intermediate Responses) in GraphRAG?

- RIRs are generated by the **LLM** based on batches of retrieved community summaries.
- Each RIR consists of:
  - **Key points** (facts, claims, insights).
  - A **score** per point (e.g., 0 to 1) reflecting its relevance to the query.
- The **LLM itself assigns these scores** through prompt instructions.
- Final selection uses:
  - **Top-N scoring points**, or
  - **Threshold filtering** (e.g., only keep points with score > 0.7).
- The selected points form the **Aggregated Intermediate Response**, used to answer the query.

## Are Community Summaries Stored in a Vector DB?

•**Yes**, summaries of each community are:

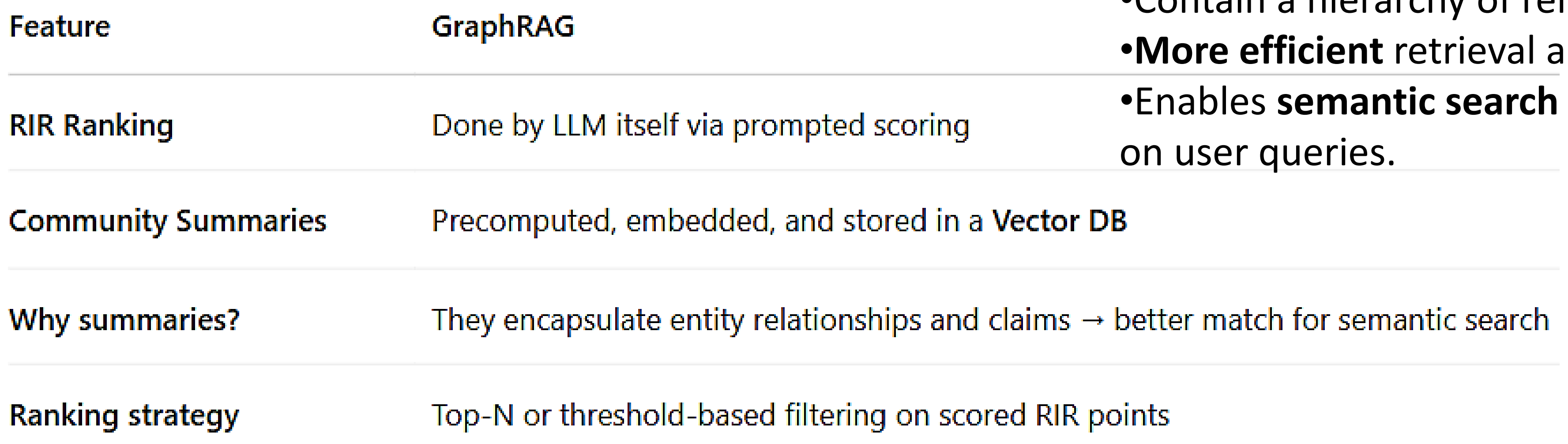
- **Generated during indexing** using LLMs.
- **Embedded** into vector representations.
- **Stored** in a **Vector Database** (e.g., FAISS, Milvus).

•These summaries include:

- Main entities
- Relationships
- Key claims from that community

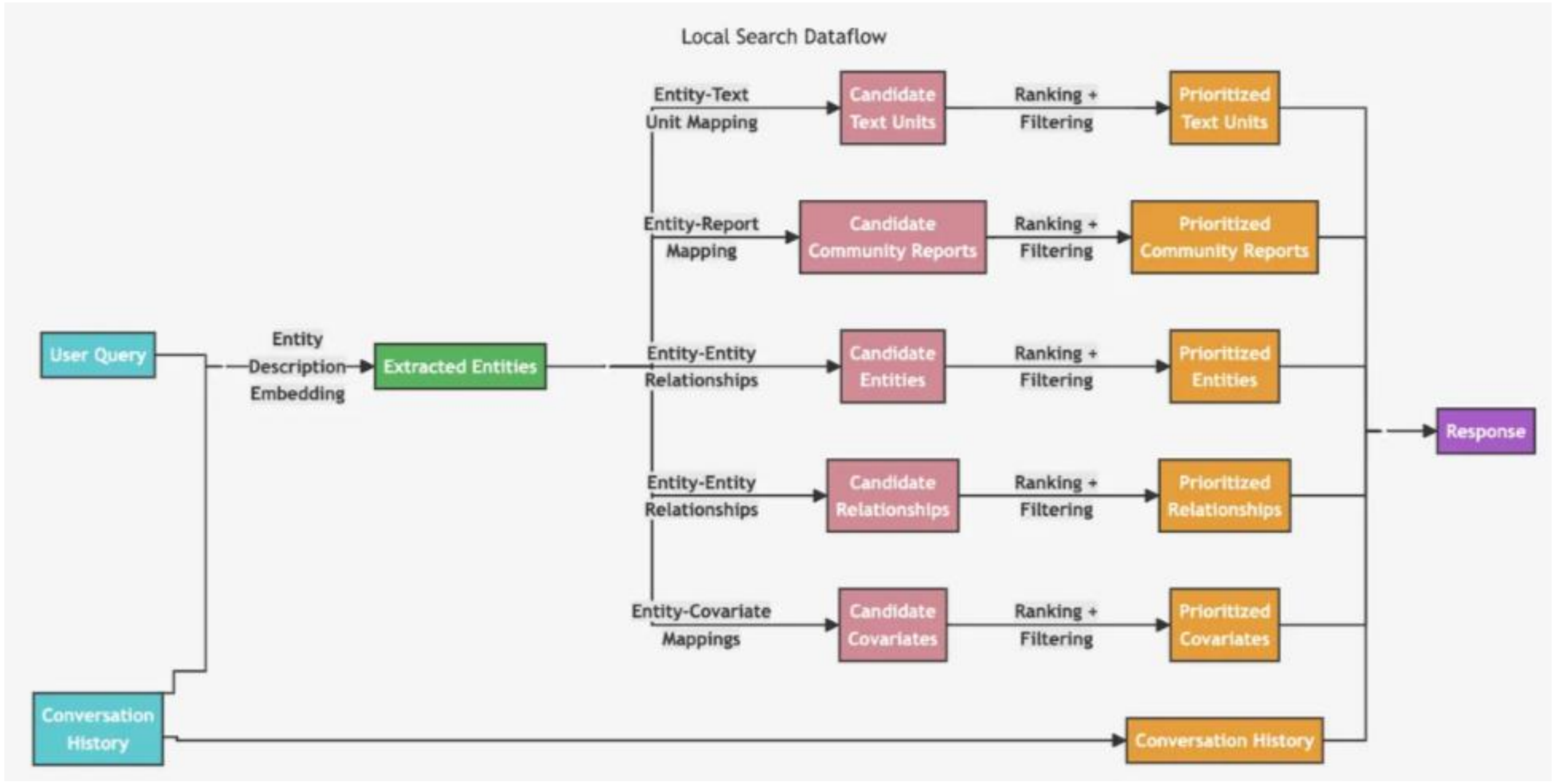
## Why Use Summaries in Vector DB Instead of Triplets?

•Summaries are **richer and more semantically meaningful**.
•Contain a hierarchy of related information
•**More efficient** retrieval and better query matching.
•Enables **semantic search** to find relevant communities based on user queries.



| Feature | GraphRAG |
|---|---|
| **RIR Ranking** | Done by LLM itself via prompted scoring |
| **Community Summaries** | Precomputed, embedded, and stored in a **Vector DB** |
| **Why summaries?** | They encapsulate entity relationships and claims → better match for semantic search |
| **Ranking strategy** | Top-N or threshold-based filtering on scored RIR points |

# Local Search – **For Specific Entity-Level Questions**



# Local Search – For Specific Entity-Level Questions

## 🔷 Use Case:

- When the query focuses on a **specific entity**, like a person, org, or concept.
- Example: *“What has OpenAI published about Reinforcement Learning?”*

## 🧠 Steps:

1. **User Query**: LLM receives the query.
2. **Entity Matching**:
   1. Using a **vector DB (e.g., Milvus)**, it identifies entities **semantically similar** to the query.
   2. But leads to a confusion, don’t we have community summaries stored in the vector DB?
3. **Entity-Text Unit Mapping**:
   1. The system retrieves the **original text units or triplets** connected to those entities, source documents, paragraphs, or sentences

## Tailoring for Local Search

•**Query-Time Hybrid Retrieval**:

- Use the vector DB to "narrow down" candidate communities.
- Leverage the graph to "zoom in" on entities within those communities.

•**Example**: For *"impact of Transformer models in NLP"*:

- Vector DB finds the "NLP Architectures" community.
- Graph retrieves the "Transformer" entity node and its connections (e.g., BERT, GPT).

Query → Vector DB → Retrieve candidate **communities** → Query knowledge graph → Resolve **entities/relationships**.

## Local Search – For Specific Entity-Level Questions

🔷 **Use Case:**

- When the query focuses on a **specific entity**, like a person, org, or concept.
- Example: *"What has OpenAI published about Reinforcement Learning?"*

🧠 **Steps:**

1. **User Query**: LLM receives the query.
2. **Entity Matching**:
    1. Using a **vector DB (e.g., Milvus)**, it identifies entities **semantically similar** to the query.
    2. But leads to a confusion, don't we have community summaries stored in the vector DB?
3. **Entity-Text Unit Mapping**:
    1. The system retrieves the **original text units or triplets** connected to those entities, source documents, paragraphs, or sentences
4. **Entity-Relationship Extraction**:
    1. Extracts relationships between entities.
    2. Example: (OpenAI, published, paperX)
5. **Covariate Mapping**:
    1. Adds extra info (attributes, stats) tied to the entity.
    2. E.g., location, dates, metrics.
6. **Community Report Integration**:
    1. Pulls in **related community summaries** for added context.
7. **Conversation Context Use**:
    1. Uses prior chat history to better understand the query's intent.

This gives a mix of **fine-grained (entity-level)** + **coarse-grained (cluster-level)** context.

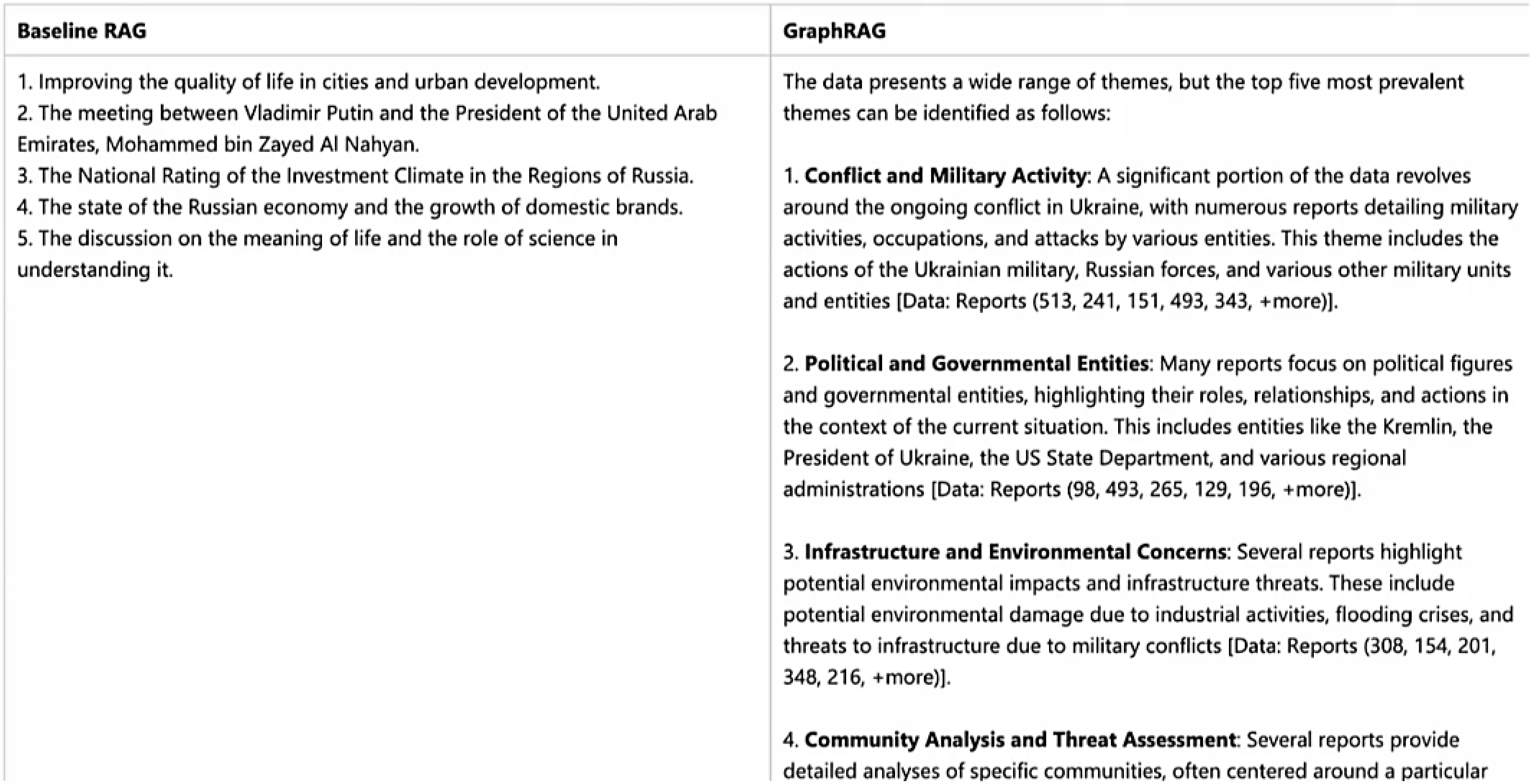
8. **Final Response**:
    1. Generates an answer based on all filtered and enriched information.

# Global vs Local

| Feature | Global Search | Local Search |
|---|---|---|
| Scope | Whole dataset | Specific entities |
| Input context | Community summaries | Entity + relationships |
| Retrieval base | Graph community hierarchy | Entity neighborhood graph |
| Method | Summarization → batch scoring → final answer | Entity fanning → relation extraction → final answer |
| Ideal for | Open-ended, high-level reasoning | Precise, entity-focused questions |
| Uses rating? | ✅ (RIRs) | ❌ (direct filtering) |

# Comparison of RAG vs GraphRAG

## Q: What are the top 5 themes in the dataset?



| Baseline RAG | GraphRAG |
|---|---|
| 1. Improving the quality of life in cities and urban development.<br>2. The meeting between Vladimir Putin and the President of the United Arab Emirates, Mohammed bin Zayed Al Nahyan.<br>3. The National Rating of the Investment Climate in the Regions of Russia.<br>4. The state of the Russian economy and the growth of domestic brands.<br>5. The discussion on the meaning of life and the role of science in understanding it. | The data presents a wide range of themes, but the top five most prevalent themes can be identified as follows:<br><br>1. **Conflict and Military Activity**: A significant portion of the data revolves around the ongoing conflict in Ukraine, with numerous reports detailing military activities, occupations, and attacks by various entities. This theme includes the actions of the Ukrainian military, Russian forces, and various other military units and entities [Data: Reports (513, 241, 151, 493, 343, +more)].<br><br>2. **Political and Governmental Entities**: Many reports focus on political figures and governmental entities, highlighting their roles, relationships, and actions in the context of the current situation. This includes entities like the Kremlin, the President of Ukraine, the US State Department, and various regional administrations [Data: Reports (98, 493, 265, 129, 196, +more)].<br><br>3. **Infrastructure and Environmental Concerns**: Several reports highlight potential environmental impacts and infrastructure threats. These include potential environmental damage due to industrial activities, flooding crises, and threats to infrastructure due to military conflicts [Data: Reports (308, 154, 201, 348, 216, +more)].<br><br>4. **Community Analysis and Threat Assessment**: Several reports provide detailed analyses of specific communities, often centered around a particular |

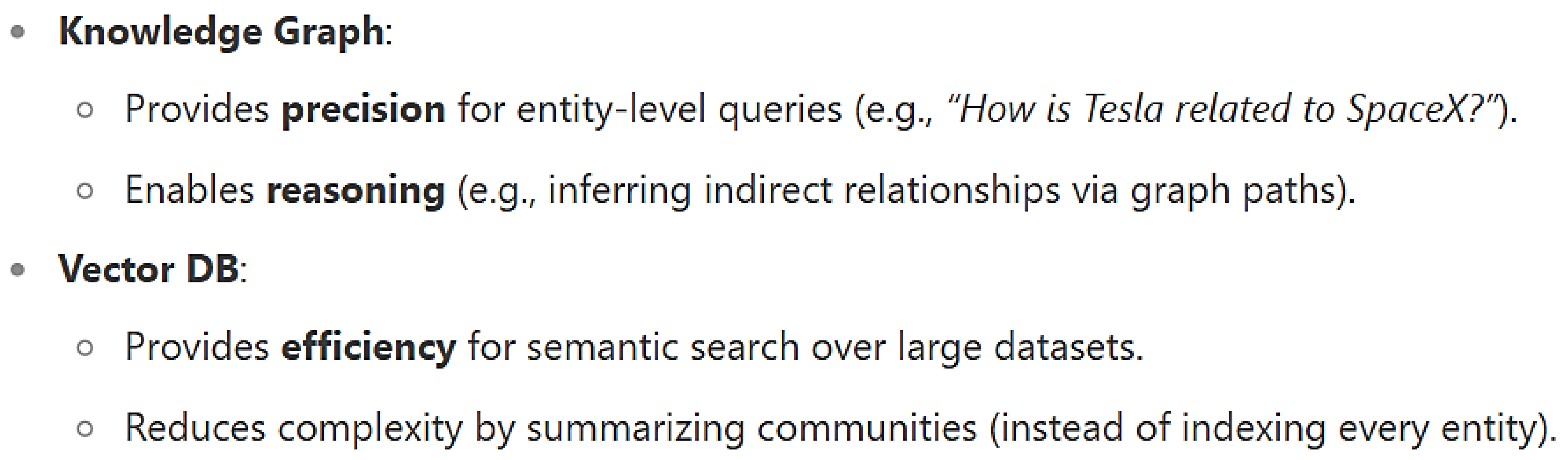
## Indexing Phase (Preprocessing)

| Step | Description |
|---|---|
| **1. Text Chunking** | Splits documents into smaller units (paragraphs/sentences). |
| **2. Triplet Extraction** | Uses LLMs to identify (subject, relation, object) triplets → knowledge graph. |
| **3. Leiden Clustering** | Detects **communities** in the graph: groups of entities strongly connected. |
| **4. Community Summaries** | Generates summaries per community containing entities, relationships, and claims. |

## Querying Phase (Runtime)

| Step | Global Search | Local Search |
|---|---|---|
| Query type | Holistic overview | Specific, focused |
| Uses community summaries? | ✅ Core input | ✅ Add-on context |
| Uses entity graph traversal? | ❌ | ✅ |
| Steps | RIRs → Rank → Aggregate → Answer | Entity match → Relationship + Covariate mapping → Answer |
| Use case | Broad trends, comparisons | Entity profiles, connections |

## • Why both knowledge graphs and vectorDB are required in GraphRAG?



- **Knowledge Graph**:
  - Provides **precision** for entity-level queries (e.g., *"How is Tesla related to SpaceX?"*).
  - Enables **reasoning** (e.g., inferring indirect relationships via graph paths).
- **Vector DB**:
  - Provides **efficiency** for semantic search over large datasets.
  - Reduces complexity by summarizing communities (instead of indexing every entity).

## Does GraphRag stores/maintains conversation history automatically?

**No — you have to explicitly manage and provide the history** as part of the system design.
GraphRAG **itself doesn't store user history automatically**.
It assumes that the **conversation history** is:
- Passed to the system at query time,
- Properly formatted and concatenated (often as a prefix or context window),
- And used by the **retrieval and/or generation components** to provide continuity.

```
Query: "Who funded companies that collaborated with OpenAI?"

→ (OpenAI, collaborated_with, Company X)
→ (Company X, funded_by, VC Firm Y)
```

**Multi hop traversal**

| | |
|---|---|
| "What are the top 5 research areas in 2024 AI papers?" | **Global** |
| "What has OpenAI published on RLHF?" | **Local** |
| "How did transformer models evolve after BERT?" | **Global** |
| "Who collaborated with OpenAI in 2022?" | **Local** |

**Global vs local real-life examples/queries**