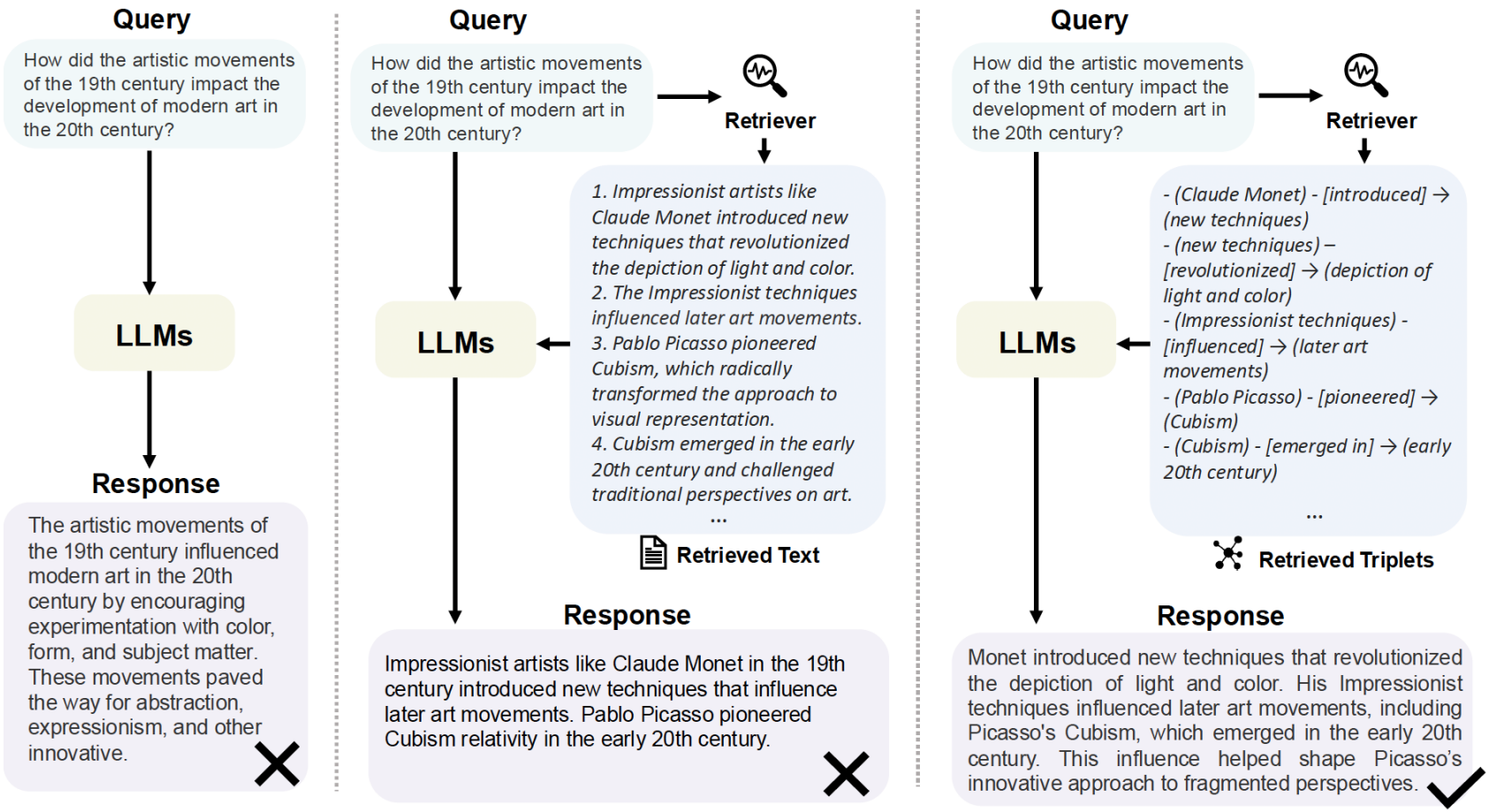
Graph RAG

Wook-Shin Han

Motivating example for Graph RAG



Text-Attributed Graphs (TAGs)

The graph data used in Graph RAG, a TAG, can be denoted as:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A}, \{x_v\}_{v \in \mathcal{V}}, \{e_{i,j}\}_{i,j \in \mathcal{E}})$$

where:

- $\mathcal{V}$: Set of nodes.
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$: Set of edges.
- $\mathcal{A} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}|}$: Adjacency matrix.
- $\{x_v\}_{v \in \mathcal{V}}$: Textual attributes of nodes.
- $\{e_{i,j}\}_{i,j \in \mathcal{E}}$: Textual attributes of edges.

One typical kind of TAG is a Knowledge Graph (KG), where:

- Nodes are entities.
- Edges are relations among entities.
- Text attributes are the names of entities and relations.

Example of Text-Attributed Graph (TAG)

Knowledge Graph (KG) Example:

- **Nodes (Entities):** {Person, City, Company}
- **Edges (Relations):** {LivesIn, WorksAt}
- **Text Attributes:**
  - Node Attribute: `"Alice"` (Person), `"New York"` (City), `"Google"` (Company).
  - Edge Attribute: `"since 2020"` (LivesIn), `"as Software Engineer"` (WorksAt).

Graph Neural Networks (GNNs)

Classical GNNs (e.g., GCN, GAT, GraphSAGE) adopt a message-passing manner to obtain node representations.

Node Representation Update:

$$h_i^{(l)} = \text{UPD}\left(h_i^{(l-1)}, \text{AGG}_{j\in\mathcal{N}(i)}\text{MSG}\left(h_i^{(l-1)}, h_j^{(l-1)}, e_{i,j}^{(l-1)}\right)\right)$$

- $\mathcal{N}(i)$: Neighbors of node i.
- **MSG:** Message function, computes the message based on:
  - The node itself ($h_i^{(l-1)}$).
  - Its neighbor ($h_j^{(l-1)}$).
  - The edge between them ($e_{i,j}^{(l-1)}$).
- **AGG:** Aggregation function that combines messages (e.g., mean, sum, max).
- **UPD:** Update function, updates the node's attributes with aggregated messages.

Example of GNN Message Passing

Graph Example:

- Nodes: {A, B, C}
- Edges: {(A, B), (A, C)}

Message Passing Steps:

- **Step 1:** Compute messages:

$$\text{MSG}(h_A, h_B, e_{A,B}) \quad \text{and} \quad \text{MSG}(h_A, h_C, e_{A,C})$$

- **Step 2:** Aggregate messages:

$$\text{AGG} = \text{mean}(\text{MSG}(A \to B), \text{MSG}(A \to C))$$

- **Step 3:** Update node representation:

$$h_A^{(l)} = \text{UPD}(h_A^{(l-1)}, \text{AGG})$$

GraphRAG I

leverages external knowledge graphs to improve contextual understanding of LMs and generate more informed responses.

Process Definition

$$a^* = \arg\max_{a \in A} p(a|q, G)$$

where a^* is the optimal answer of the query q given the TAG G, and A is the set of possible responses.

Joint Modeling and Answer Generation

$$p(a|q, G) = \sum_{G' \subseteq G} p_\theta(a|q, G') p_\theta(G'|q, G) \approx p_\theta(a|q, G^*) p_\theta(G^*|q, G),$$

where G^* represents the optimal subgraph that provides the most relevant information.

GraphRAG II

Role of Graph Retriever and Answer Generator

- The graph retriever identifies the optimal subgraph G^*, focusing on efficiency and relevance
- The answer generator then uses this subgraph G^* to compute and generate the optimal answer a^*, applying learned parameters for precision.

Graph-Based Indexing

Graph-Based Indexing is the initial phase of GraphRAG. It focuses on identifying or constructing a graph database G aligned with downstream tasks and establishing indices on it.

Sources of Graph Data:

- Public knowledge graphs
- Proprietary data sources:
  - Textual data
  - Other forms of data
- Those constructed based on proprietary data

Indexing Process:

- Mapping node and edge properties
- Establishing pointers between connected nodes
- Organizing data for fast traversal and retrieval operations

Importance: Indexing defines the granularity of the retrieval stage

Graph-Guided Retrieval (G-Retrieval)

Given a user query q, expressed in natural language, it aims to extract the most relevant elements (e.g., entities, triplets, paths, subgraphs) from knowledge graphs.

Formulation

$$
\begin{aligned}
G^* &= \text{G-Retriever}(q, G) \\
&= \arg \max_{G \subseteq \mathcal{R}(G)} p_\theta(G|q, G) \\
&= \arg \max_{G \subseteq \mathcal{R}(G)} \text{Sim}(q, G),
\end{aligned}
$$

- G^* is the optimal retrieved graph elements
- $\text{Sim}(\cdot, \cdot)$: similarity function between q and retrieved graph
- $\mathcal{R}(\cdot)$: A function to narrow the search range of subgraphs

Graph-Enhanced Generation (G-Generation)

synthesizes meaningful outputs or responses based on the retrieved graph data

Formulation

$$
\begin{aligned}
a^* &= \text{G-Generator}(q, G^*) \\
&= \arg\max_{a \in A} p_\phi(a|q, G^*) \\
&= \arg\max_{a \in A} p_\phi(a|\mathcal{F}(q, G^*)),
\end{aligned}
$$

- G^*: Retrieved graph elements
- $\mathcal{F}(\cdot, \cdot)$: Converts graph data into a form the generator can process

Open Knowledge Graphs

General Knowledge Graphs

- **Encyclopedic Knowledge Graphs:** E.g., Wikidata, Freebase, DBpedia, and YAGO
- **Commonsense Knowledge Graphs:** E.g., ConceptNet and ATOMIC

Domain Knowledge Graphs

Specialized knowledge in particular fields, such as:

- **Biomedical:** E.g., CMeKG and CPubMed-KG
- **Entertainment:** E.g., Wiki-Movies
- **Various Domains:** E.g., GR-Bench and GraphQA

Self-Constructed Graph Data

enables the customization and integration of proprietary or domain-specific knowledge into retrieval systems.

Purpose and Application

construct graphs from varied sources like documents, tables, and other databases.

Examples and Methods

- **Document Relationships:** A heterogeneous document graph captures relations such as co-citation and co-topic.
- **Passage Connections:** Establish relationships based on shared keywords among passages.
- **Entity Relations:** Utilize NER tools and language models to extract entities and their relations, forming a knowledge graph.

Indexing Methods in Graph Data Processing

Graph Indexing

Preserves the entire structure of the graph, ensuring easy access to all edges and neighboring nodes.

Text Indexing

Converts graph data into textual descriptions stored in a text corpus. Employs text-based retrieval techniques

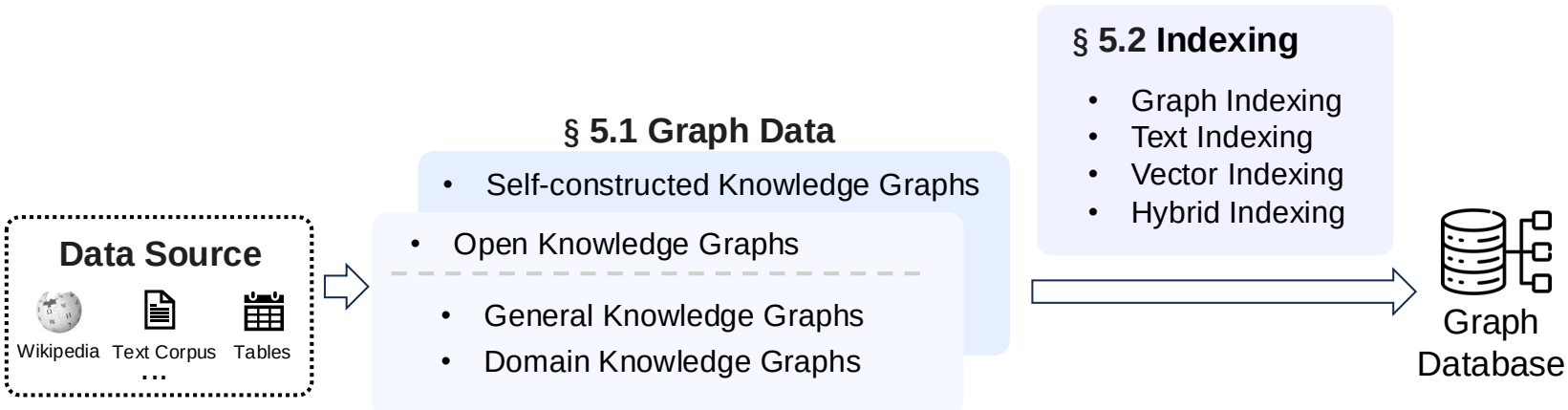
Vector Indexing

Transforms graph data into vector representations to enhance retrieval efficiency using k-NN such as LSH.

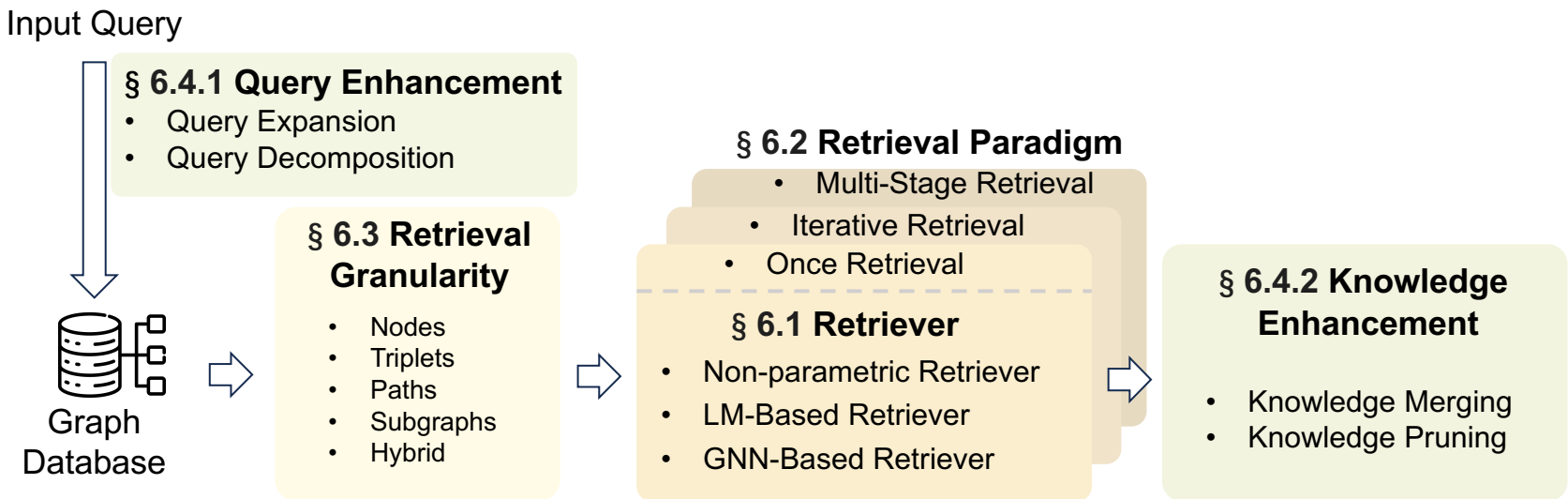
Hybrid Indexing

Combines graph, text, and vector indexing methods to leverage the advantages of each

The overview of graph-based indexing



The general architectures of graph-based retrieval



Retrievers in GraphRAG

- Non-parametric
- LM-based
- GNN-based

Non-parametric Retriever

Non-parametric retrievers utilize heuristic rules or traditional graph search algorithms, achieving high retrieval efficiency without deep-learning models. Examples include:

- First, use entity linking to identify nodes before retrieval
- Then, retrieve k-hop paths containing topic entities or
- use Enhancing the Prize-Collecting Steiner Tree (PCST) algorithm for optimal subgraph extraction.

Retrieving k-hop paths containing topic entities

QA Context

A **revolving door** is convenient for **two direction travel**, but also serves as a **security measure** at what?

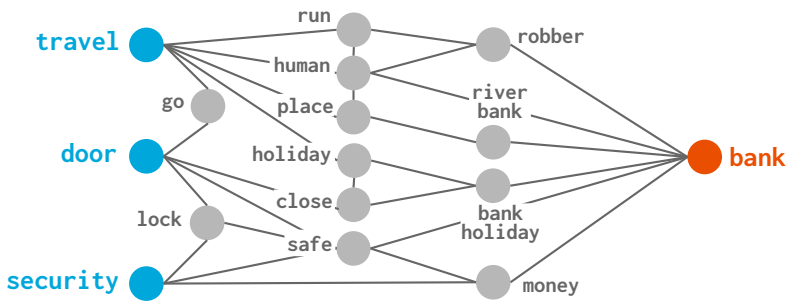
A. **bank*** B. library C. department store
D. mall E. new york

Language Model

Relevance(entity | QA context)

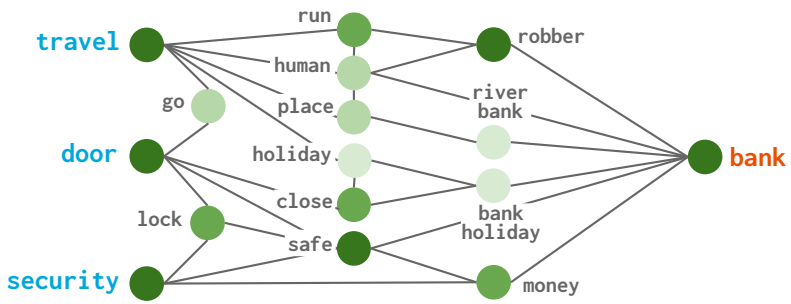
entity

Retrieved KG



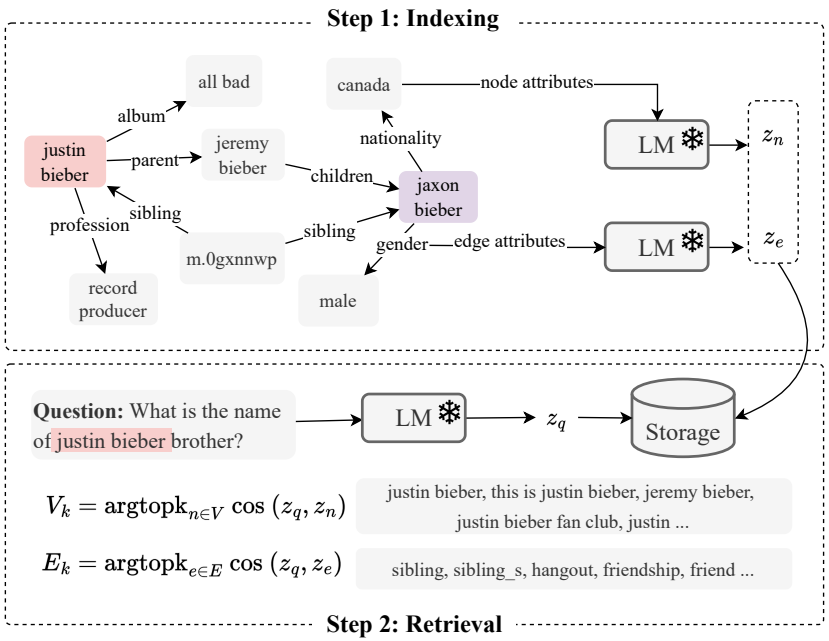
Some entities are more relevant than others given the context.

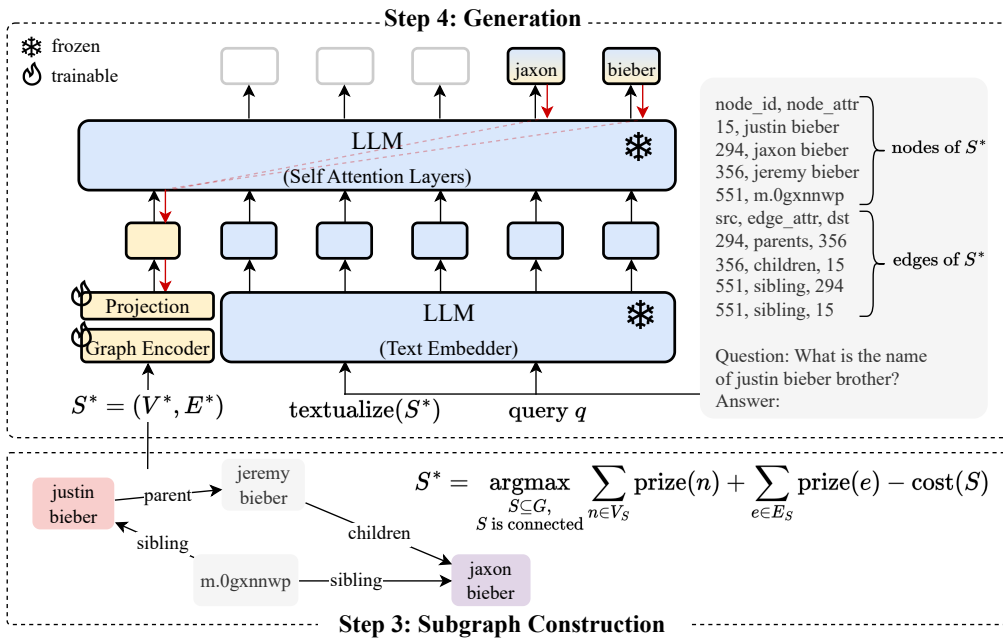
KG node scored



Entity relevance estimated. **Darker** color indicates higher score.

Overview of G-Retrieval

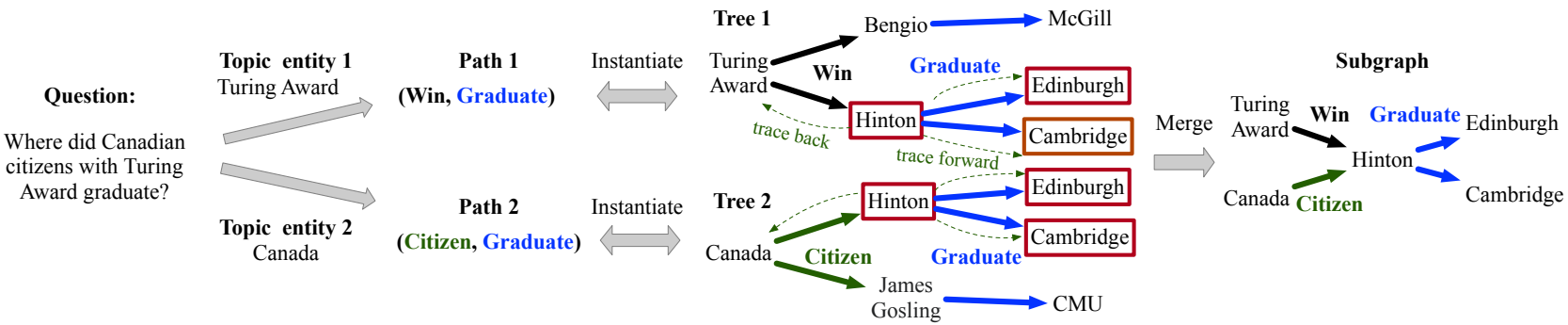




LM-based Retriever

- Subgraph Retriever: Exploit RoBERTa as the retriever, expanding from the topic entity and retrieving relevant paths in a sequential decision process.
- KG-GPT: Adopts LLMs to generate the set of top-K relevant relations of the specific entity.
- StructGPT: Utilizes LLMs to automatically invoke several pre-defined functions such as Extract_Neighbor_Relations, retrieving and combining relevant information to assist further reasoning.

Subgraph Retriever



GNN-based Retriever

GNNs effectively leverage graph structures, encoding graph data to score and retrieve entities or paths relevant to queries:

- Encoding graphs and assigning scores to entities.
- Iteratively retrieving relevant paths using LLaMA2 and GNN embeddings.
- GNN-RAG: Encodes the graph, assigns a score to each entity, and retrieves entities relevant to the query based on a threshold.
- EtD: Iterates multiple times to retrieve relevant paths. During each iteration:
  - Uses LLaMA2 to select edges connecting the current node.
  - Employs GNNs to obtain embeddings of the new layer of nodes for the next round of LLM selection.

Discussion

Discussing retrieval efficiency and accuracy:

- Non-parametric methods are efficient but may suffer from inaccuracy.
- LM and GNN retrievers provide higher accuracy but require more computational resources.
- Hybrid approaches and multi-stage strategies improve both efficiency and accuracy.

Retrieval Paradigm

Within GraphRAG, different retrieval paradigms, including once retrieval, iterative retrieval, and multi-stage retrieval, play crucial roles in improving the relevance and depth of the retrieved information:

- Once retrieval: Gathers all pertinent information in a single operation.
- Iterative retrieval: Conducts further searches based on previously retrieved information, progressively narrowing down to the most relevant results. Divided into:
  - Adaptive retrieval: Model determines when to stop retrieval.
  - Non-adaptive retrieval: Stops after a fixed number of iterations.
- Multi-stage retrieval: Divides retrieval into multiple stages using different retrievers at each stage.

Once Retrieval

Once retrieval aims to retrieve all relevant information in a single query:

- Embedding-based approaches: Use similarity to retrieve relevant pieces of information
- Rule-based methods: Extract specific structured information like triplets or subgraphs
- Multiple independent retrievals: E.g., instruct LLMs to generate reasoning paths, then retrieve matching subgraphs in parallel.

Iterative Retrieval

Iterative retrieval employs multiple steps, where subsequent searches depend on prior results:

- Non-adaptive retrieval:
  - PullNet: Retrieves problem-relevant subgraphs through fixed iterations.
  - KGP: Updates context and retrieves based on similarity.
- Adaptive retrieval:
  - Uses LMs for hop prediction
  - Employs termination signals like special tokens or retrieval tools

Discussion

- Once retrieval
  - Lower complexity and shorter response times.
  - Suitable for real-time scenarios.
- Iterative retrieval
  - Higher time complexity due to iterative refinement.
  - Yields higher retrieval accuracy.
- Choice of retrieval paradigm should balance accuracy and time complexity based on use cases.

Retrieval Granularity

Retrieval granularity is determined by the form of related knowledge retrieved from graph data, tailored to different task scenarios:

- Nodes: Retrieve individual entities or textual attributes.
- Triplets: Retrieve subject-predicate-object relationships.
- Paths: Retrieve sequences of relationships between entities.
- Subgraphs: Retrieve comprehensive relational contexts.
- Hybrid: Combine multiple granularities for enhanced relevance.

Nodes

Nodes provide precise retrieval for targeted queries:

- Focus on individual elements or entities in the graph.
- Examples:
  - Retrieve passage nodes
  - Retrieve entities from knowledge graphs

Triplets

Triplets provide structured representations of relationships:

- Subject-predicate-object format for relational data.
- Examples:
  - Retrieve triplets containing topic entities.
  - Convert triplets into sentences for retrieval.
  - Generate logical chains and retrieve related triplets.

Paths

Paths enhance contextual understanding and reasoning:

- Capture sequences of relationships between entities.
- Examples:
  - Traverse to find paths within n-hop.
  - HyKGE: Retrieve paths, co-ancestor chains, and co-occurrence chains.
  - ToG: Perform beam search to find reasoning paths.
  - GNN-RAG: Extract paths satisfying length relationships.

Subgraphs

Subgraphs capture comprehensive relational contexts:

- Extract patterns, sequences, and dependencies.
- Examples:
  - Form subgraphs using two-hop neighbors.
  - Encode k-hop ego networks for retrieval.
  - Extract evidence subgraphs based on rules.
- Combine paths to induce subgraphs

Hybrid Granularities

Hybrid approaches combine multiple granularities:

- Enhance detail and contextual understanding.
- Examples:
  - Use LLM agents to adaptively select nodes, triplets, paths, and subgraphs.

Discussion

Key considerations in retrieval granularity:

- Granularities overlap: Subgraphs include paths, and paths consist of triplets.
- Task-dependent granularity selection:
  - Finer granularities (nodes, triplets) for efficiency and relevance.
  - Hybrid approaches for comprehensive understanding in complex tasks.
- Flexibility enables adaptation across diverse domains.