

GRAG: Graph Retrieval-Augmented Generation

Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, Liang Zhao

Department of Computer Science

Emory University

Atlanta, GA 30322, USA

{yuntong.hu, liang.zhao}@emory.edu

Presenter: Joohyung Yun

2024.11.13

Recap

Retrieval Augmented Generation

Motivation: Traditional language models rely only on *parametric knowledges* which are

- hard to fine-tune
- hard to explain (e.g., why is this output generated?)

Idea: *Retrieve, then generate*

- A *retriever* retrieves query-relevant documents from a corpus (i.e., database containing text)
- A *reader* generates the output based on the query and the retrieved results

In this class, we have focused on **two aspects of RAG**

- How can we retrieve query-relevant documents? How can we use them to generate an answer?
- What are different modalities of retrieving documents? How should we modify our retrieval and reading method depending on the modality?

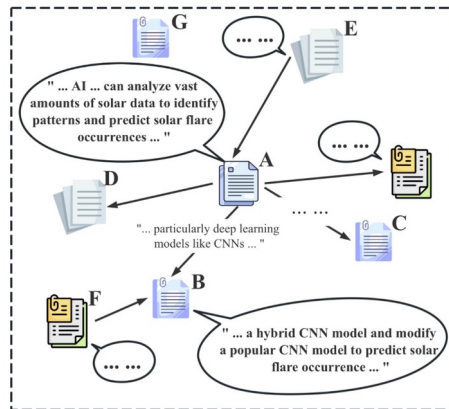
Introduction Textual Graphs

In recent years, the correlations between documents are heavily researched.

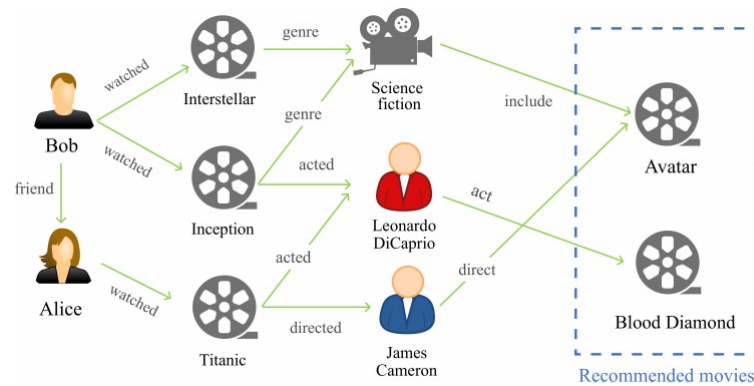
They can be expressed in the format of *textual graphs* where

- Nodes express documents
- Edges express the relationships between two documents

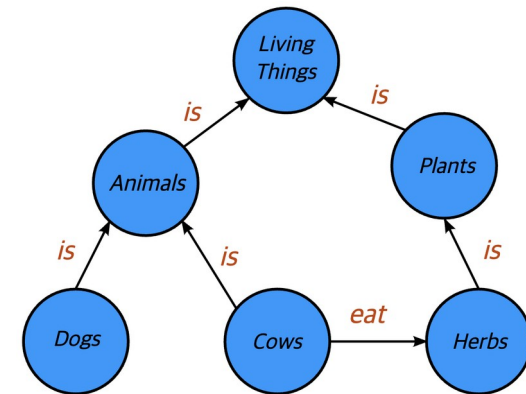
Some use cases of textual graphs:



Citation graph



Recommender system



Knowledge graph

Figure 1. Example use cases of textual graphs

Problem Description

Textual Graph Question Answering

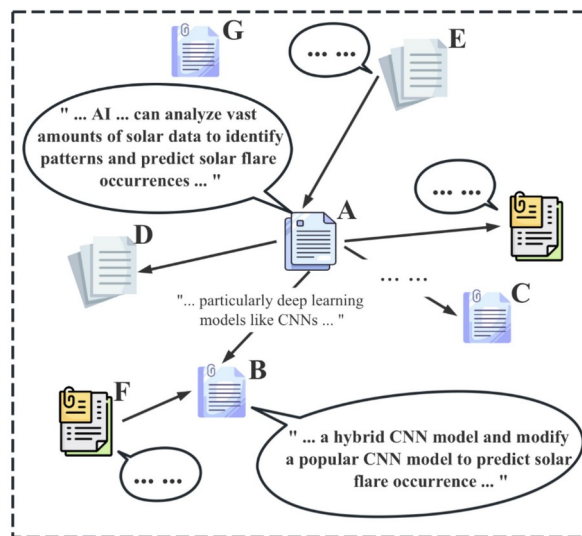
Given a query q and a textual graph G , generate the correct answer a .

- Query $q = [q_1, \dots, q_t]$: A sequence of tokens
- Textual graph G : (to be defined in the next page)
- Answer $a = [a_1, \dots, a_r]$: A sequence of tokens

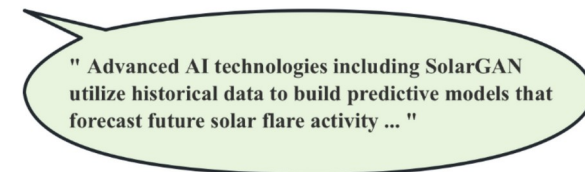
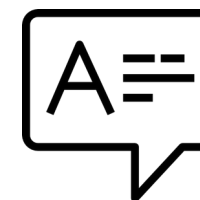
"How can AI's recent progress advance solar flare prediction?"



Query q



Textual graph G



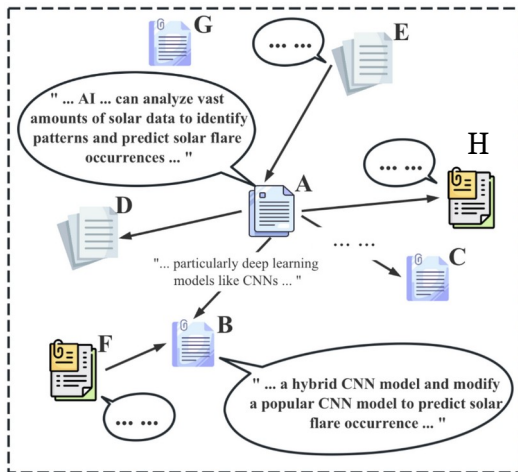
Answer a

Notation Definition

Textual Graph

$\mathcal{G} = (V, E, \Gamma, \Lambda)$: A node-labeled, edge-labeled graph

- V : Set of nodes
- E : Set of edges
- Γ : A function that maps a node to its attribute
- Λ : A function that maps an edge to its attribute



$$V = \{A, B, C, D, E, F, G, H\}$$

$$E = \{(A, B), (A, C), (A, D), (A, H), (E, A), (F, B)\}$$

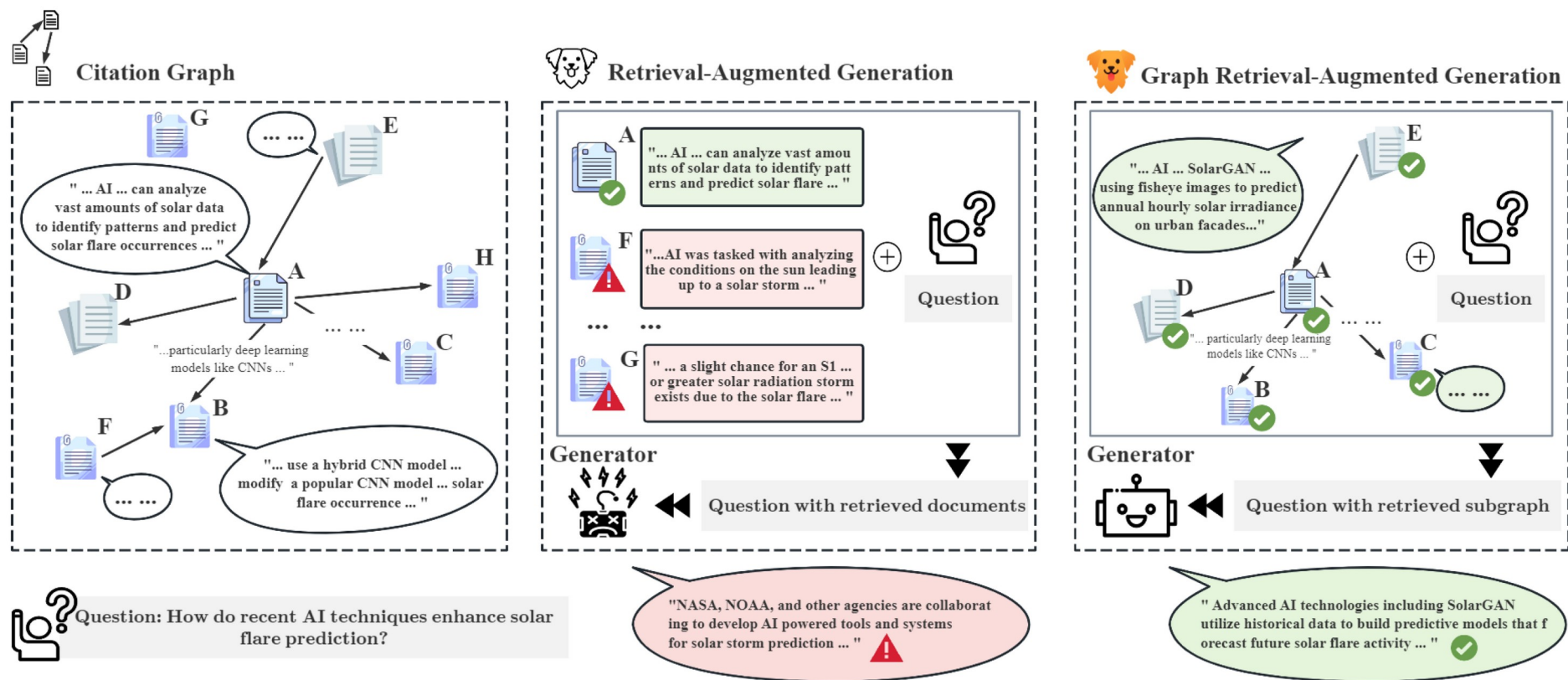
$$\Gamma = \{(A \rightarrow \text{"... AI... can analyze vast amounts of solar data ..."}), (B \rightarrow \text{"... a hybrid CNN model and modify a popular CNN ..."}), \dots\}$$

$$\Lambda = \{((A, B) \rightarrow \text{"... particularly deep learning models..."}), \dots\}$$

Textual Graph RAG Retrieves a Textual Subgraph

Textual graph RAG problem first retrieves a subgraph from the original textual graph.

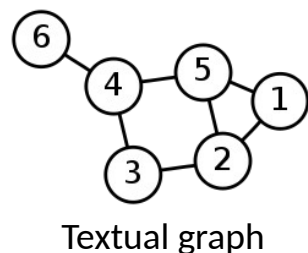
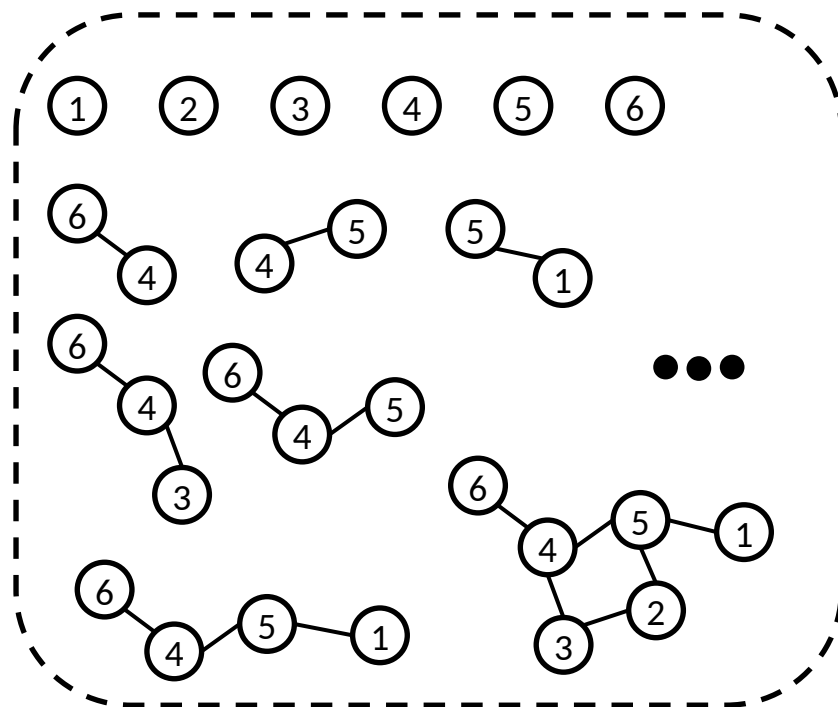
- The relationships between documents must be maintained after the retrieval.



Two Challenges of GRAG

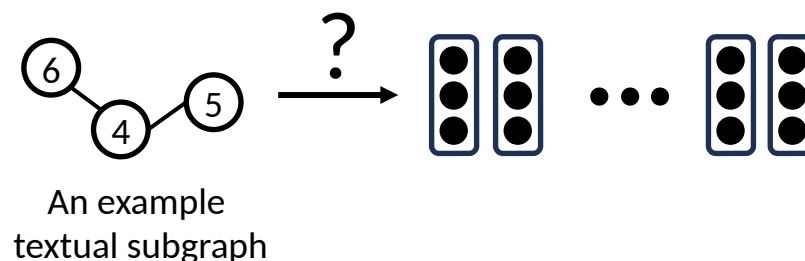
1. **Challenge #1 (Retriever):** How can we *efficiently retrieve query-relevant textual subgraph*?
 - Naïvely enumerating all possible subgraphs results in $2^{|V|+|E|}$ cases of subgraphs.
2. **Challenge #2 (Reader):** How should we deliver the textual subgraph's *textual (document-based) information and topological (relation-based) information jointly to the LLM*?

Challenge #1

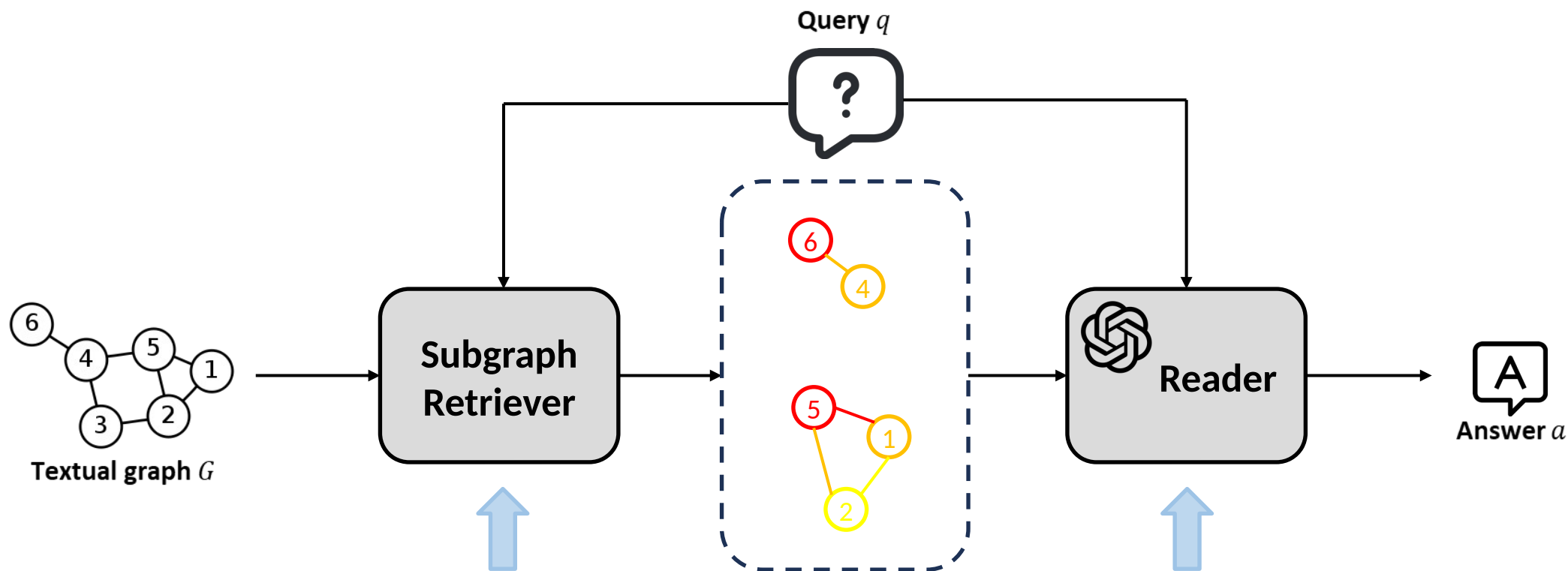


Set of all possible textual subgraphs

Challenge #2



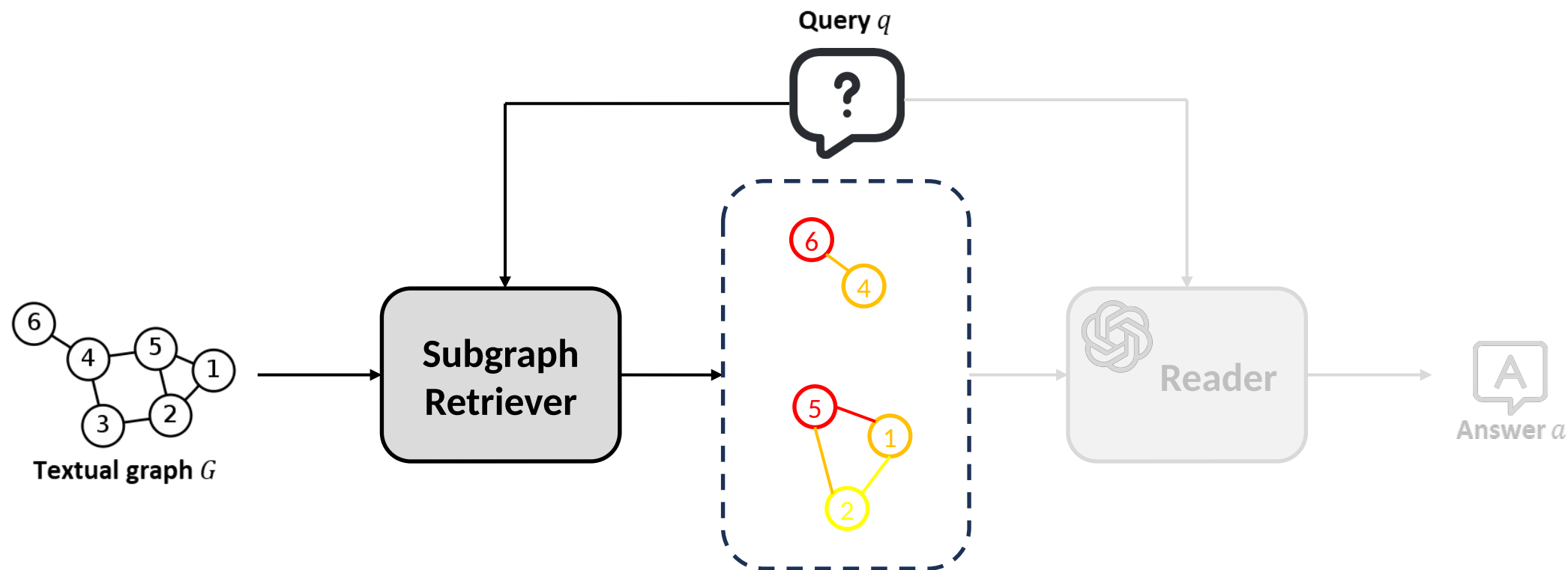
Overview of GRAG



Questions:

- How did the subgraph retriever solve challenge #1?
- How did the reader solve challenge #2?

Overview of GRAG



Questions:

- How did the subgraph retriever solve challenge #1?
- How did the reader solve challenge #2?

Challenge of GRAG's Retriever

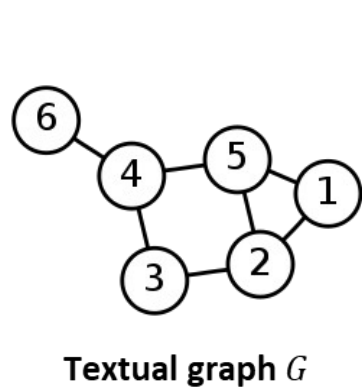
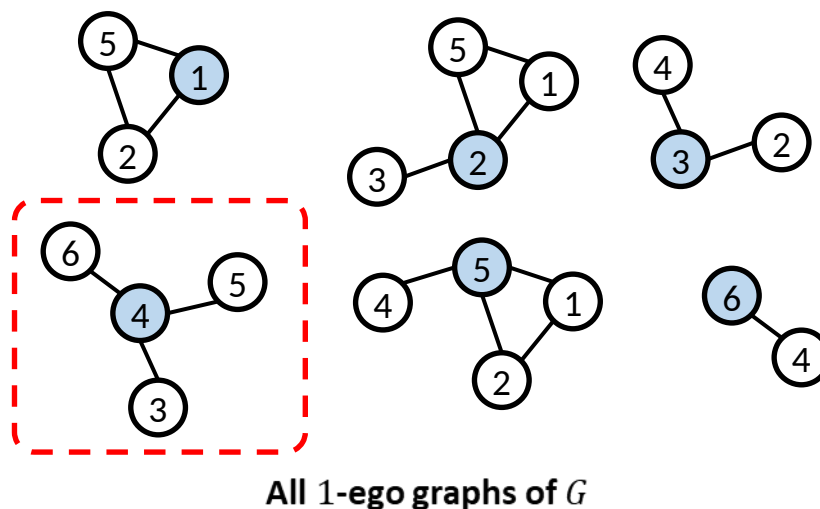
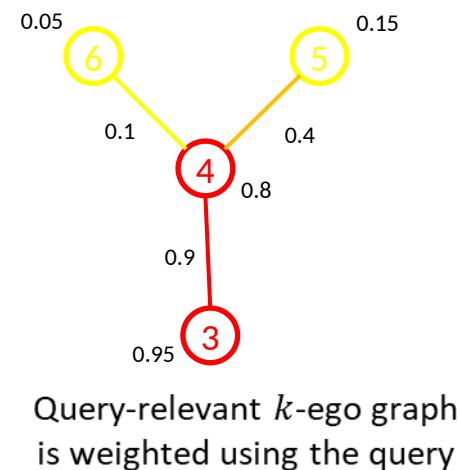
Challenge #1: How can we *efficiently retrieve query-relevant textual subgraph G* ?

- Naïvely enumerating all possible subgraphs results in $2^{|V|+|E|}$ cases of subgraphs.

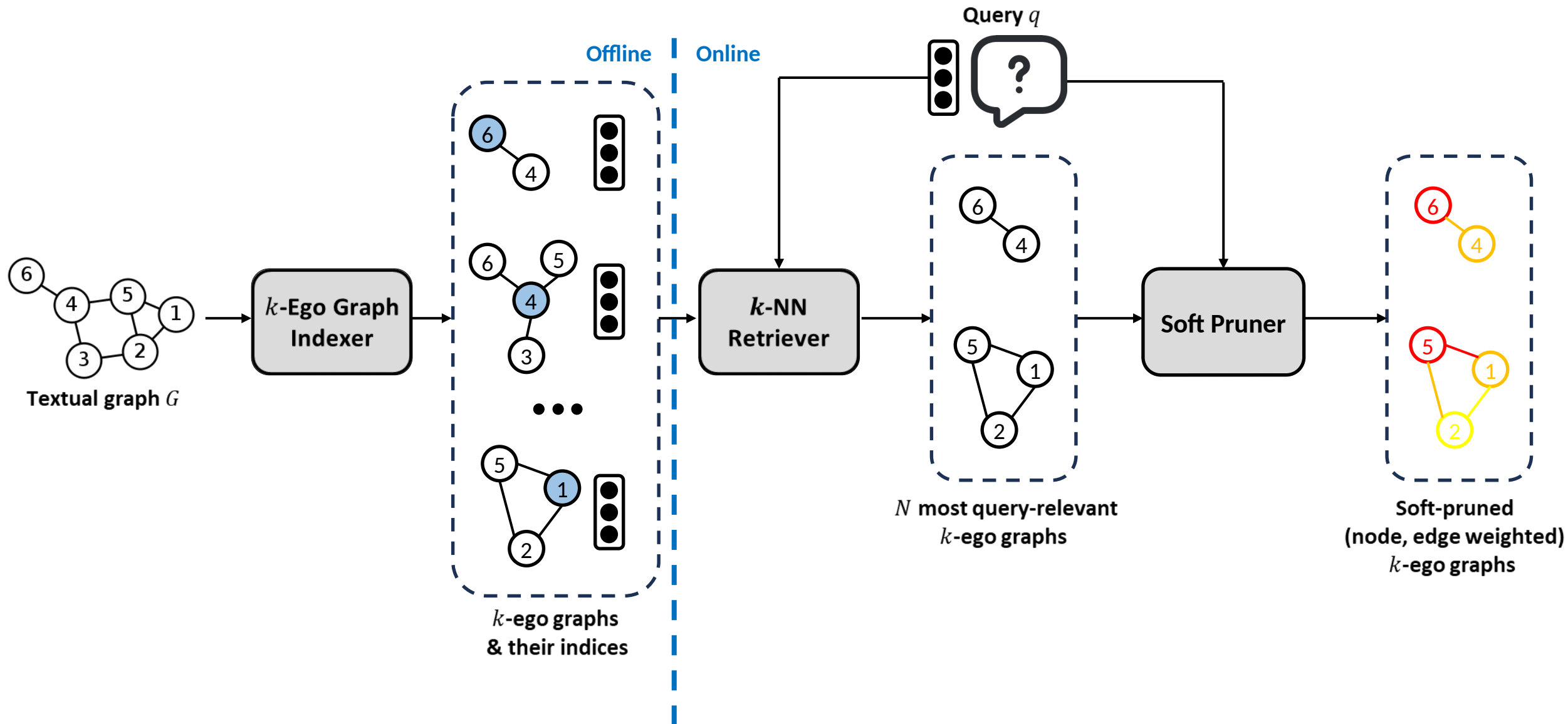
Motivation (Assumption): A query-relevant subgraph has a form of a *graph with a center node and its related nodes within k hops*.

Idea: Reduce the retrieval candidates by only considering $|V|$ approximate candidates.

- (Offline) Embed every $|V|$ k -ego graphs of G
 - k -ego graph of G : A subgraph of G where a node of G becomes the center node, and is composed of nodes reachable within k hops
- (Online) Pick N query-relevant k -ego graphs
- (Online) Weight the nodes & edges of N retrieved candidates with the relevant to the query.



GRAG's Retriever



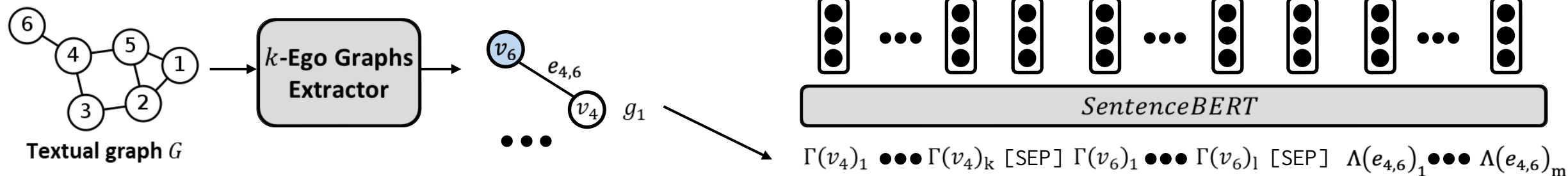
Retriever *k*-ego Graph Index Generation

Input: G (Textual graph)

1. Extract $|V|$ k -ego graphs $g_1, \dots, g_{|V|}$ of G
2. Generate index z_g for each k -ego graph g using a pretrained language model (SentenceBERT)

$$z_g = \text{POOL}(\text{SentenceBERT}(\Gamma(v_1); \dots; \Gamma(v_n); \Lambda(e_1); \dots; \Lambda(e_m)))$$

- Γ : A function that maps a node to its attribute
- Λ : A function that maps an edge to its attribute



Retriever

Query-relevant k -ego Graph Retrieval

Input: $z_{g_1}, \dots, z_{g_{|V|}}$ (Indices for k -ego graphs)

Input: q (Query)

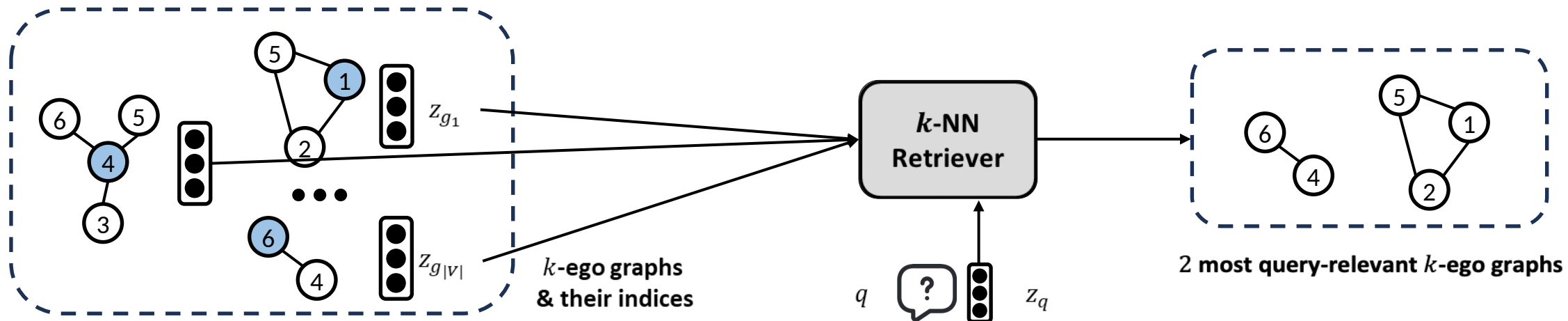
Output: $\hat{g}_1, \dots, \hat{g}_N$

1. Embed the query using a pretrained language model (SentenceBERT).

$$z_q = \text{SentenceBERT}(q)$$

2. Perform a simple k -NearestNeighbor search to find the top N query-relevant k -ego graphs.

$$\{\hat{g}_1, \dots, \hat{g}_N\} = \underset{g \in S(G)}{\operatorname{argmax}} N \cos(z_q, z_g)$$



Retriever

Soft Pruning (Node & Edge Weighting)

Input: $\hat{g}_1, \dots, \hat{g}_N$ (Top N query-relevant k -ego graphs)

Input: q (Query)

Output: $\hat{g}_1', \dots, \hat{g}_N'$ (Input k -ego graphs with their nodes & edges weighted)

Motivation:

- k -ego graphs were formed independent to the query and may contain irrelevant nodes or edges.
- Figure out query-relevant nodes within the k -ego graphs at on-line time using the query information.

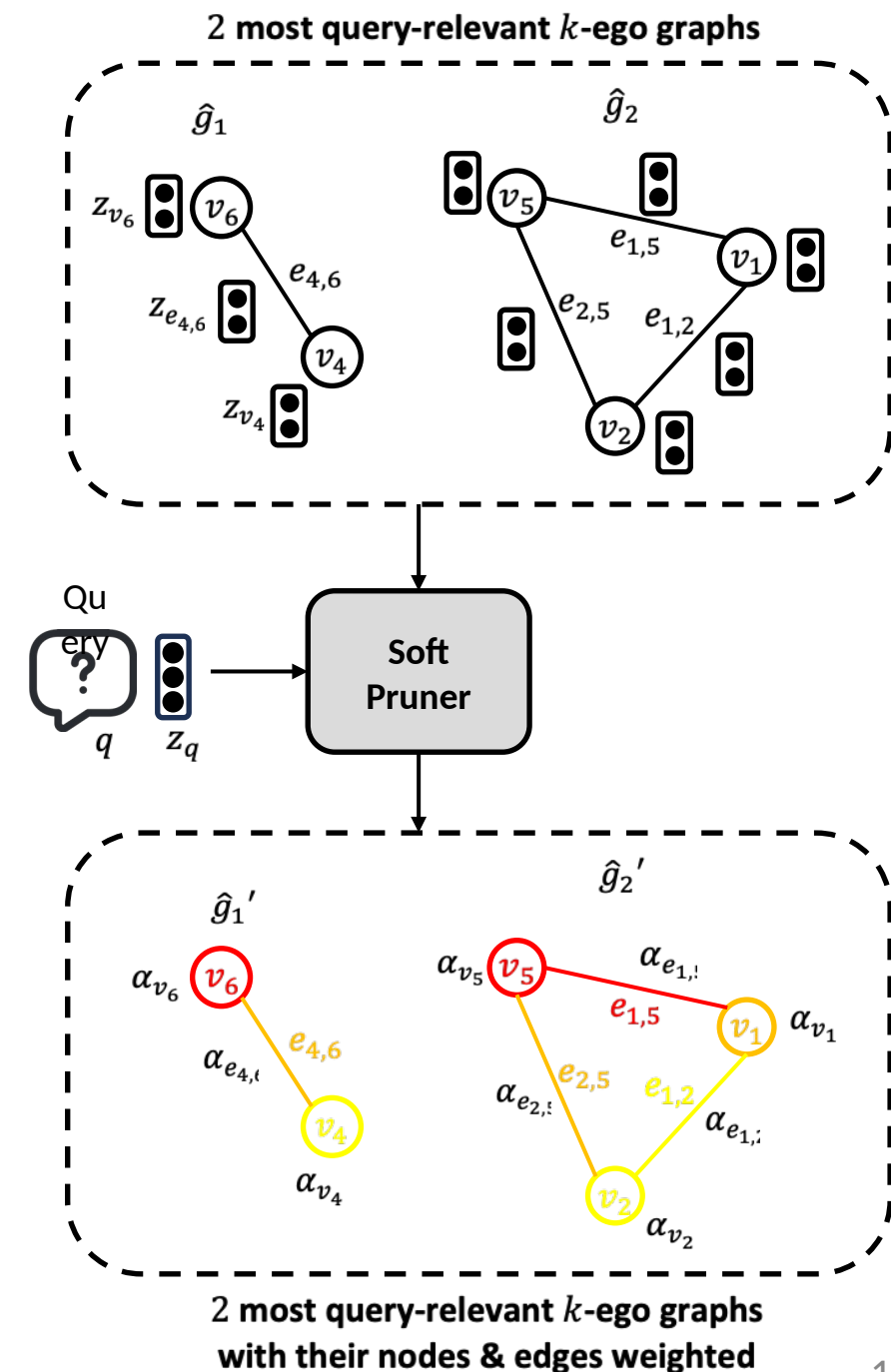
1. Weight each node v in graph $\hat{g}$ using the query q .

$$\alpha_v = MLP_{\phi_1}(z_v \ominus z_q)$$

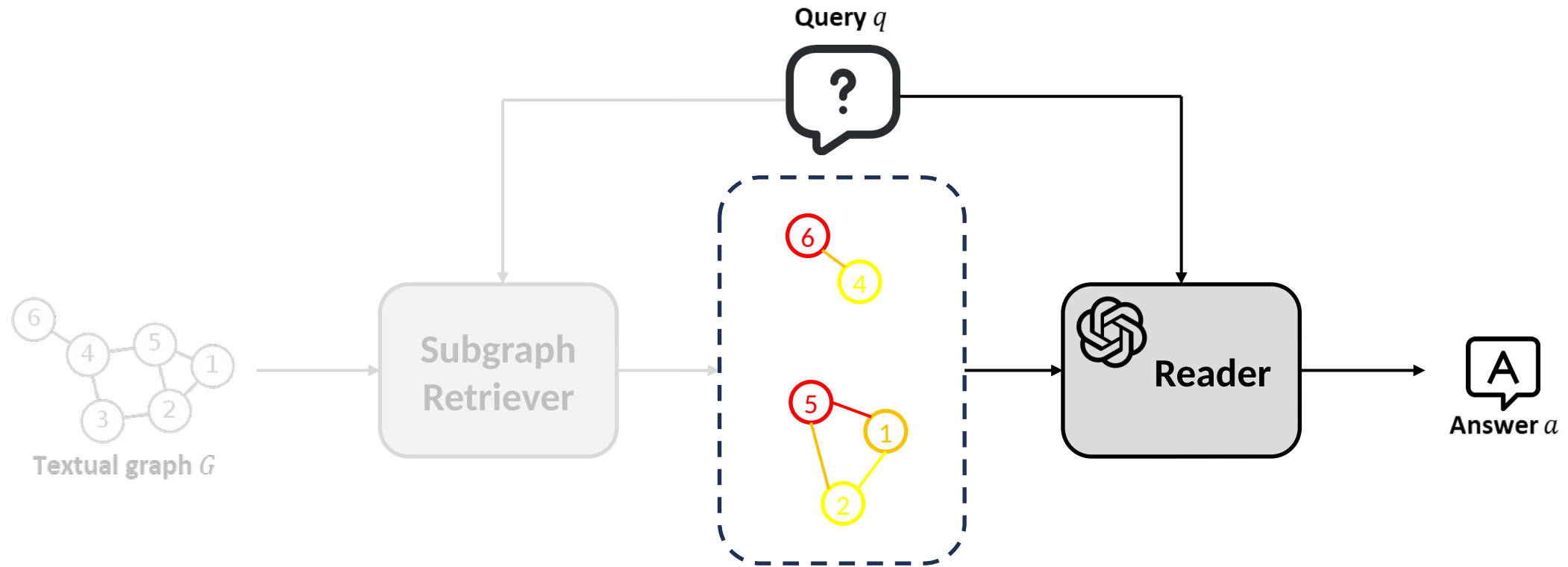
2. Weight each edge e in graph $\hat{g}$ using the query q .

$$\alpha_e = MLP_{\phi_2}(z_e \ominus z_q)$$

$$a \ominus b = \sqrt{\sum_{i=1}^d (a[i] - b[i])^2}$$



Overview of GRAG (Revisited)



Questions:

- How did the subgraph retriever solve challenge #1?
- How did the reader solve challenge #2?

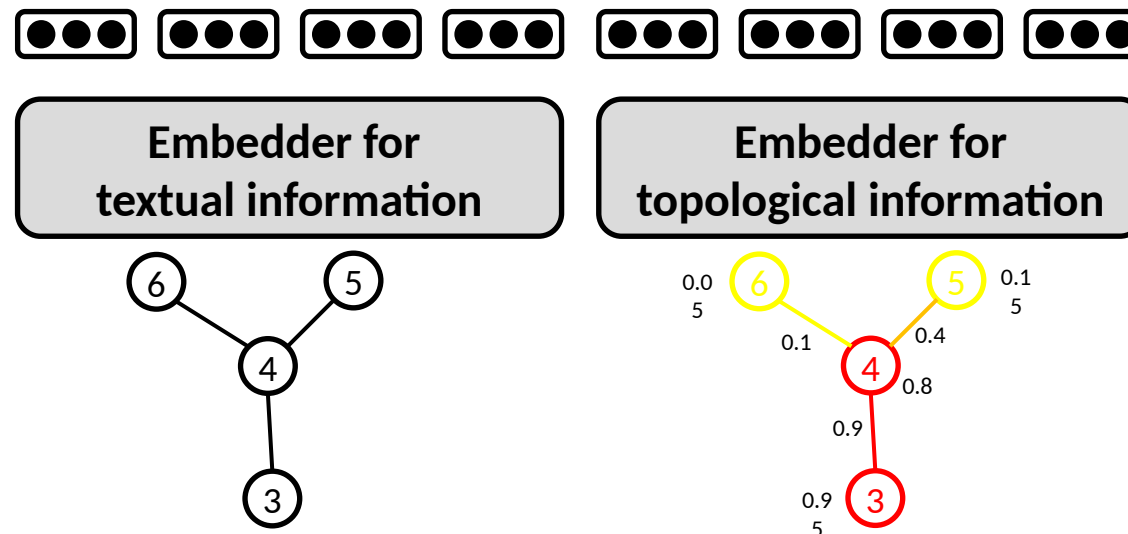
Challenge of GRAG's Reader

Challenge #2: How should we deliver a retrieved textual subgraph's *textual (document-based) information* and *topological (relation-based) information* jointly to the LLM?

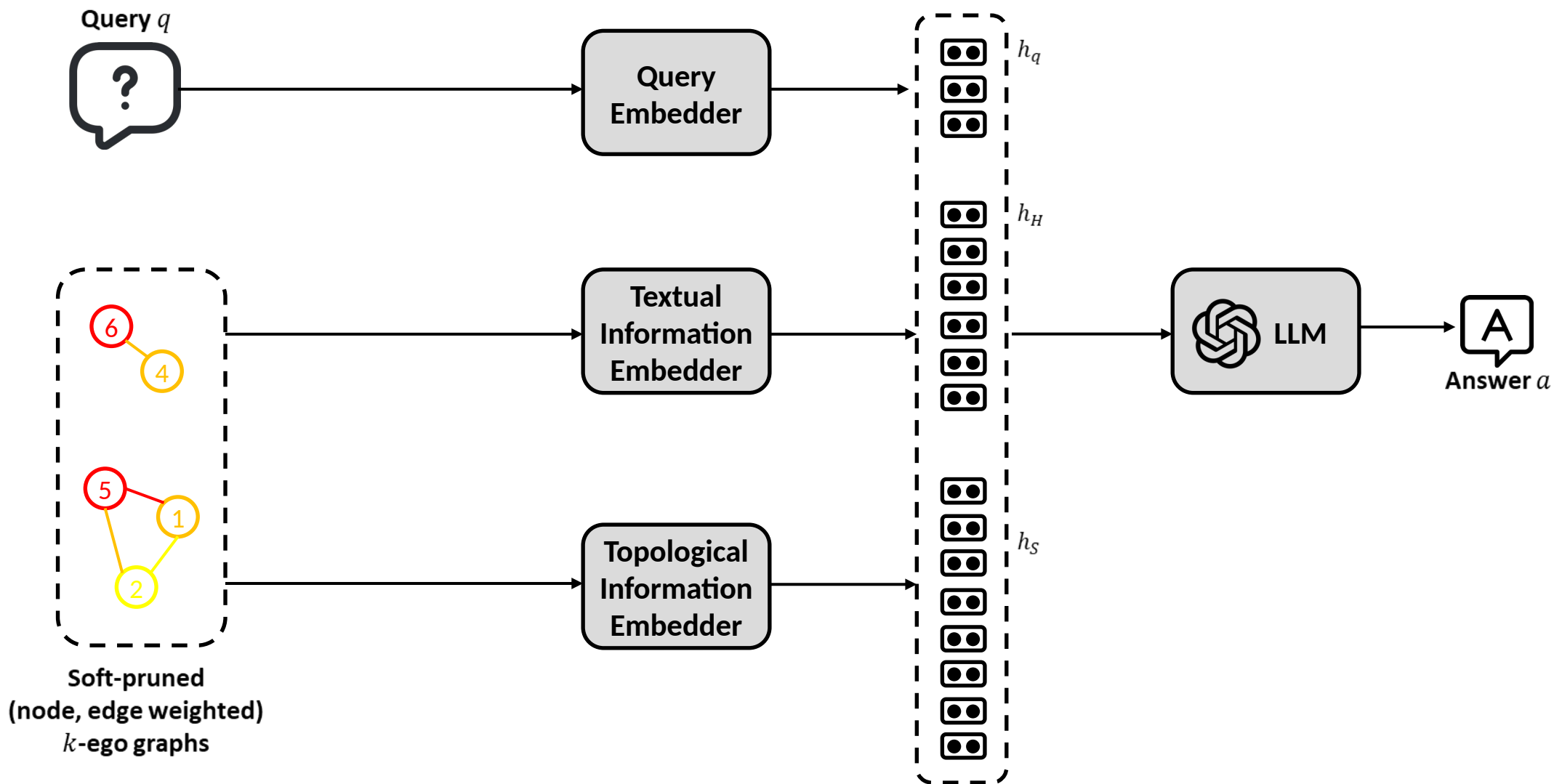
Motivation: Use two embedders that are specialized for each kind of information.

Idea:

- (Hard prompting) Embed the textual information of the retrieved subgraphs
- (Soft prompting) Embed the topological information of the retrieved subgraphs



GRAG's Reader



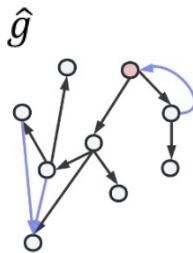
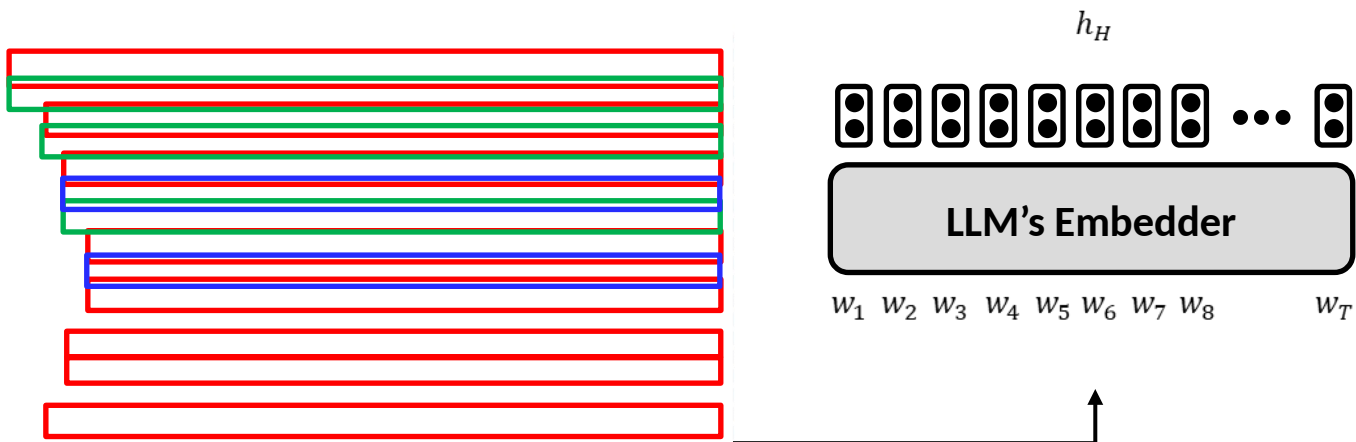
Reader Hard Prompting

Goal: Accurately embed the textual information within a retrieved graphs

Input: $\hat{g}_1, \dots, \hat{g}_N$ (retrieved subgraphs)

Output: $h_{H_1}, \dots, h_{H_N}$

1. Express k -ego graph in a tree structure.
 - Generate a spanning tree of a k -ego graph by performing a breadth-first traversal starting from the center.
2. Translate the spanning tree into a natural language prompt.
 - Express each node v with its text mapping $\Gamma(v)$
 - Express each edge e using (i) hierarchical relationship between two nodes and (ii) its text mapping $\Lambda(e)$.
 - Express cycles in natural language.
3. Embed the prompt with LLM's embedder.



Reader Soft Prompting

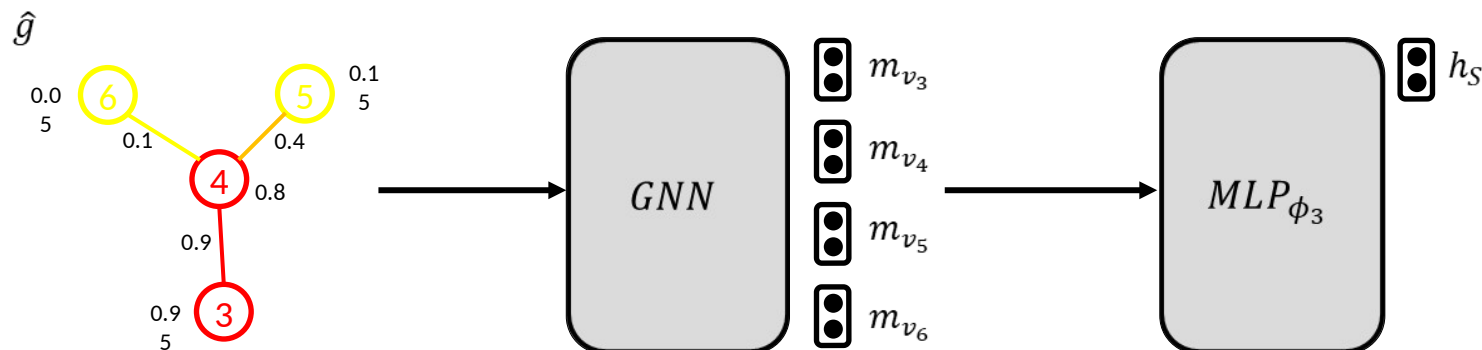
Goal: Accurately embed the topological information within a retrieved graphs

Input: $\hat{g}_1', \dots, \hat{g}_N'$ (retrieved weighted subgraphs)

Output: $h_{S_1}, \dots, h_{S_N}$

1. Pass a retrieved graph $\hat{g}$ through a learnable graph neural network.
2. Pass the graph neural network's output to a learnable MLP to align it to LLM's embedding space.

$$h_S = MLP_{\phi_3}(GNN(\hat{g}))$$



Experiments

Goals of Experiment

1. **Accuracy comparison:** Does GRAG accurately answer the textual graph questions?
2. **Parameter sensitivity:** How sensitive is GRAG's accuracy regarding its two parameters?
 - k : The number of hops for k -ego graphs
 - N : Number of query-relevant k -ego graphs to retrieve
3. **Ablation study:** How does each of GRAG's submodule effect the end-to-end accuracy?

Experiment Settings

Dataset

- WebQSP
- ExplaGraphs

Metrics

- Recall
- F1
- Hit@1
- Exact Match

Competitors

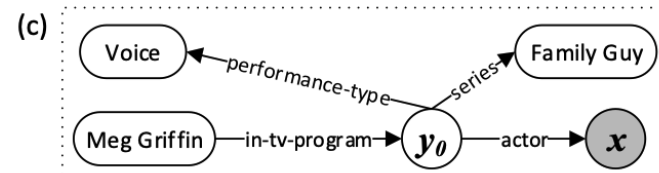
- **RAG (Retriever)**
 - BM25
 - MiniLM-L12-v2
 - LaBSE
 - mContriever
 - E5
 - G-Retriever
- **LLMs**
 - Llama-7b
 - Llama-7b + Fine-tuning

Table 1: Dataset statistics (average).

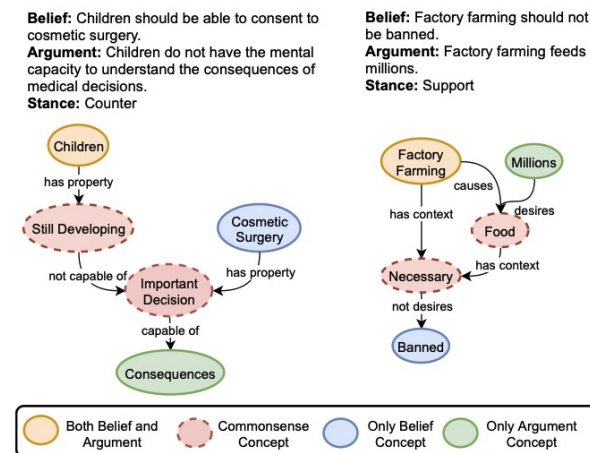
Dataset	WebQSP	ExplaGraphs
# Graphs	4,700	2,766
# Nodes	1370.89	5.17
# Edges	4252.37	4.25
# Tokens	100,627	1,396

(a) “who voiced meg on family guy?”

(b) Topic Entity: Meg Griffin (m.035szd)
Inf. Chain: in-tv-program – actor
Constraints: (1) y_0 – series – Family Guy (m.019nnl)
(2) y_0 – performance-type – Voice (m.02nsjvf)



Example of WebQSP



Example of ExplaGraphs

Accuracy Comparison

Table 2: Performance comparison across WebQSP and ExplaGraphs datasets. **Bold** numbers indicate the best performance among all models. **Highlight** numbers demonstrate the performance improvement achieved by our GRAG approach compared to the LLM baselines.

Model	Prompt tuning	Fine-tuning	WebQSP			ExplaGraphs
			F_1 Score $\uparrow$	Hit@1 $\uparrow$	Recall $\uparrow$	Acc $\uparrow$
Baselines						
LLM only	$\times$	$\times$	0.2555	0.4148	0.2920	0.3394
LLM _{LoRA}	$\times$	$\checkmark$	0.4295	0.6186	0.4193	0.8927
Compared Retrievers						
BM25	$\times$	$\times$	0.2999	0.4287	0.2879	0.6011
MiniLM-L12-v2	$\times$	$\times$	0.3485	0.4730	0.3289	0.6011
LaBSE	$\times$	$\times$	0.3280	0.4496	0.3126	0.6011
mContriever-Base	$\times$	$\times$	0.3172	0.4453	0.3047	0.5866
E5-Base	$\times$	$\times$	0.3421	0.4705	0.3254	0.6011
G-Retriever	$\checkmark$	$\times$	0.4674	0.6808	0.4579	0.8825
G-Retriever _{LoRA}	$\checkmark$	$\checkmark$	0.5023	0.7016	0.5002	0.9042
Our Retrieval Approach						
GRAG	$\checkmark$	$\times$	0.5022	0.7236	0.5099	0.9223
Δ_{LLM}			$\uparrow 96.56\%$	$\uparrow 74.45\%$	$\uparrow 74.62\%$	$\uparrow 171.74\%$
GRAG _{LoRA}	$\checkmark$	$\checkmark$	0.5041	0.7275	0.5112	0.9274
Δ_{LoRA}			$\uparrow 17.37\%$	$\uparrow 17.60\%$	$\uparrow 21.92\%$	$\uparrow 3.89\%$

GRAG outperforms other RAGs and LLM baselines for both datasets.

- Fine-tuning offers only marginal performance gains in GRAG.

Parameter Sensitivity

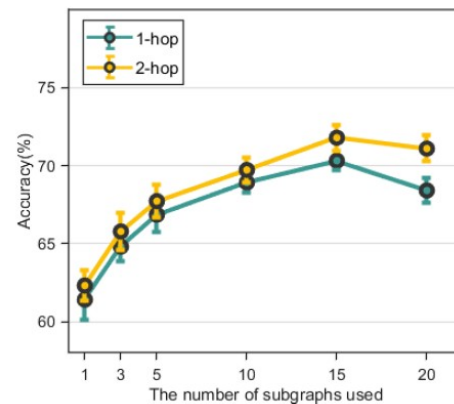


Figure 3: Performance of our GRAG approach on WebQSP as the ego-graph size and number of ego-graphs used vary.

k : The number of hops for k -ego graphs

- The end-to-end accuracy increased as k increased from 1 to 2.
- As k increases, the overall efficiency is lowered due to longer training and inference time.

N : Number of query-relevant k -ego graphs to retrieve

- Accuracy tends to increase until N is smaller than 15.
- Afterwards, the accuracy drops due to LLM's hallucinations from hard negatives.

Ablation Study

Table 4: Ablation study on WebQSP. Δ_{GRAG} represents the change in Hit@1 performance relative to our full GRAG approach.

Setting	Hit@1	Δ_{GRAG}
w/o Retrieval	0.6093	↓ 16.25%
w/o Graph Encoder	0.5835	↓ 19.79%
w/o Soft Pruning	0.5671	↓ 22.05%
w/o Graph Descriptions	0.4496	↓ 38.20%

- **w/o Retrieval:** Uses only parametric knowledge of fine-tuned LLM
- **w/o Graph Encoder:** Does not use soft prompting in the reader
- **w/o Soft Pruning:** Does not weight node & edges in the retriever
- **w/o Graph Descriptions:** Does not use hard prompting in the reader

Weird point: Removing the retriever results in better results than using it partially.

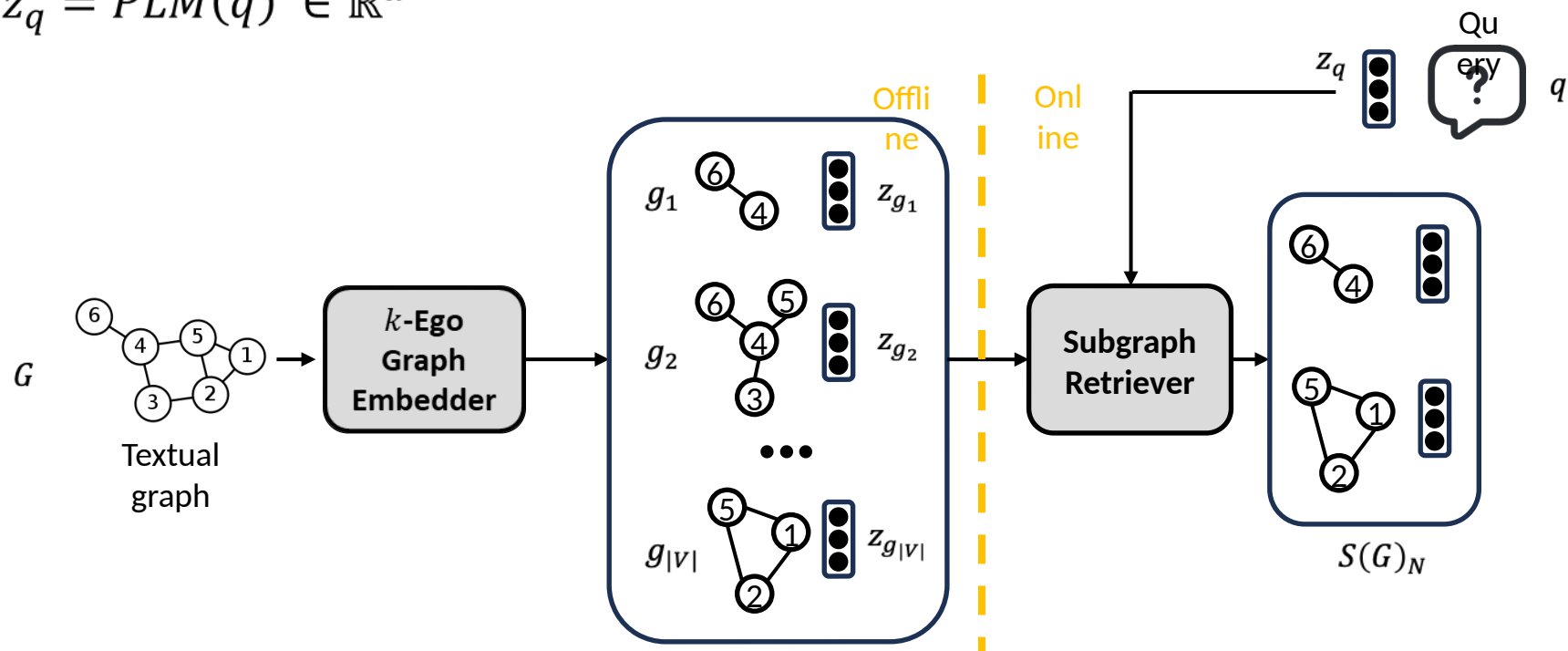
APPENDIX

Retriever *k*-ego Graph Embedding Generation

Retrieval equation:
$$S(G)_N = \operatorname{argmax}_{g \in \mathcal{S}(G)} N \operatorname{sim}(z_q, z_g)$$

k-ego graph embedding:
$$z_g = \operatorname{POOL} \left(\operatorname{PLM} \left(\{T_n\}_{n \in V_g}, \{T_e\}_{e \in V_g} \right) \right)$$

Query embedding:
$$z_q = \operatorname{PLM}(q) \in \mathbb{R}^d$$



Problem Definition

Textual Graph Retrieval-Augmented Generation (GRAG)

Given a question q over G , find an optimal subgraph structure $g \subseteq S(G)$, and using it correctly answer the question.

$$\operatorname{argmax}_{g \in S(G), X_q, X_g} p_{\theta}(Y|[X_q; X_g]) = \prod_{i=1}^n p_{\theta}(y_i|y_{<i}, [X_q; X_g])$$

Equation 1. Formal equation of GRAG.

- $y_{<i}$: prefix tokens
- X_q, X_g : token sequence of the question and integrated graph information.
- $[;]$: concatenation operator

