GROUNDEDKG-RAG: Grounded Knowledge Graph Index for Long-document Question Answering

Tianyi Zhang[1] **Andreas Marfurt**[2]
[1] getAbstract [2] Lucerne University of Applied Sciences and Arts
Switzerland
tianyiz0423@gmail.com, andreas.marfurt@hslu.ch

Abstract

Retrieval-augmented generation (RAG) systems have been widely adopted in contemporary large language models (LLMs) due to their ability to improve generation quality while reducing the required input context length. In this work, we focus on RAG systems for long-document question answering. Current approaches suffer from a heavy reliance on LLM descriptions resulting in high resource consumption and latency, repetitive content across hierarchical levels, and hallucinations due to no or limited grounding in the source text. To improve both efficiency and factual accuracy through grounding, we propose GROUNDEDKG-RAG, a RAG system in which the knowledge graph is explicitly extracted from and grounded in the source document. Specifically, we define nodes in GROUNDEDKG as entities and actions, and edges as temporal or semantic relations, with each node and edge grounded in the original sentences. We construct GROUNDEDKG from semantic role labeling (SRL) and abstract meaning representation (AMR) parses and then embed it for retrieval. During querying, we apply the same transformation to the query and retrieve the most relevant sentences from the grounded source text for question answering. We evaluate GROUNDEDKG-RAG on examples from the NarrativeQA dataset and find that it performs on par with a state-of-the art proprietary long-context model at smaller cost and outperforms a competitive baseline. Additionally, our GROUNDEDKG is interpretable and readable by humans, facilitating auditing of results and error analysis.

Keywords: knowledge graphs, retrieval-augmented generation, question answering, semantic role labeling, abstract meaning representation

1. Introduction

Retrieval-Augmented Generation (RAG) is a technique that retrieves and incorporates additional information from an external knowledge source to answer user queries more accurately. Its application scenarios are diverse. For example, a chatbot may retrieve up-to-date information released after the model's training cutoff to address users' questions, or it may search over a confidential contract provided by the user to verify specific terms and conditions.

In this work, we focus on the second scenario, where a long-form document (books with potentially millions of words) is provided, and comprehensive, accurate, and low-latency responses grounded in the document are required.

Current approaches (Sarthi et al., 2024; Edge et al., 2025) heavily rely on LLM generations to describe book chunks, entities, relations. They then create a heirarchical tree structure, recursively summarizing clusters of lower-level entities. This reliance on LLM outputs can backfire in two ways. First, recursive summarization introduces a significant risk of hallucination, as generated summaries may not be strictly grounded in the original text and can introduce content that does not exist in the source document. Second, repeated information across hierarchical levels leads to inefficiencies in both retrieval and prompt construction. Moreover, a fixed tree structure cannot adapt to different queries that may require aggregating different subsets or perspectives of the original document.

To address these issues, we propose GROUNDEDKG-RAG[1], a RAG system in which every component of the constructed knowledge graph is explicitly grounded in the original text, and aggregation is query-adaptive rather than fixed. We define a GROUNDEDKG whose nodes represent entities and actions, and whose edges encode semantic relations between entities and actions defined by PropBank (Palmer et al., 2005), as well as temporal relations between actions. With this clear definition and construction, we only need to embed the nodes while preserving the explicit graph structure. Each node embedding incorporates not only the node itself but also its local graph context including the neighboring nodes.

At query time, we retrieve nodes relevant to the user query and use the graph structure to identify semantically and structurally related nodes. The corresponding grounded text spans are then retrieved and filtered using vector-based semantic similarity and passed to a generative LLM for final answer generation. The LLM used for answer generation is exchangeable as it does not depend on our retrieval component. We use the same model

[1]The source code is available upon request.

Proceedings of the Knowledge Graphs and Large Language Models Workshop (KG-LLM) @ LREC26, pages 63–72
May 16, 2026.

as our baseline for a fair comparison.

The proposed GroundedKG-RAG framework avoids hallucinations introduced by recursive LLM summarization at multiple hierarchical levels. Every step in the pipeline is human-readable and verifiable due to explicit grounding in the source text. Furthermore, the framework enables efficient and flexible aggregation of document content tailored to different query requirements.

We evaluate GroundedKG-RAG on the NarrativeQA benchmark using Exact Match, Sequence Match, BERTScore (Zhang et al., 2020), and ROUGE-L F1 (Lin, 2004). Experimental results show that GroundedKG-RAG performs on par with a state-of-the art proprietary long-context model at smaller cost, and outperforms the GraphRAG (Edge et al., 2025) baseline across all metrics.

2. Related Work

Several recent approaches have explored question answering over long documents using retrieval-augmented generation (RAG). One line of work aims to extract node–edge pairs to build a graph index for RAG systems. LightRAG (Guo et al., 2025), HippoRAG (Gutiérrez et al., 2024) and HippoRAGv2 (Gutiérrez et al., 2025) create a knowledge graph from entities and relations in the text. However, these nodes and edges are not clearly defined and contain information at different levels, ranging from words and phrases to paragraphs and passages, making the structure redundant and difficult to organize. Moreover, during retrieval, these methods expand the found nodes with neighboring entities to enable multi-hop reasoning, a common shortcoming of retrieval-augmented generation models (Li et al., 2024).

Another line of work relies on LLMs to generate descriptions of document chunks and organize them hierarchically. In RAPTOR (Sarthi et al., 2024), leaf nodes correspond to sentences or fixed-size chunks (e.g., 100 tokens). These nodes are recursively embedded and clustered using semantic similarity, and parent-node summaries are generated by an LLM from their child nodes. These summaries are then used for retrieval. This approach relies heavily on querying LLMs to build summaries, which can lead to redundancy and hallucinations. Another side effect is the use of a fixed tree structure, which may be suboptimal when handling diverse downstream questions.

GraphRAG (Edge et al., 2025) integrates knowledge graph construction from the first line of work with the hierarchical structures used in the second approach. During the indexing stage, it constructs a hierarchical entity-level knowledge graph by prompting large language models (LLMs) to extract entities from the text. These entities are grouped into communities with the Leiden algorithm (Traag et al., 2019). The communities are then summarized by LLMs. This process of clustering and summarizing is repeated for a predefined number of levels (1–4 in the paper). In the querying stage, GraphRAG performs a vector search on the embeddings of entities, relations, and communities. It can extend the results with related entities, or even generate new sub questions (in their *drift search*) to expand the search results.

GraphRAG is a compute-intensive and weakly grounded[2] method that relies heavily on LLM generations for its search index. These shortcomings motivated our GroundedKG-RAG, which focuses on grounding the knowledge graph in the source text and is created in a resource-efficient manner.

3. GroundedKG-RAG

3.1. GroundedKG Definition

To construct a reliable knowledge graph for reasoning that is readable for humans, we define a knowledge graph G as a *directed graph*, where nodes represent entities and actions extracted from the unstructured text, and edges represent semantic relations between them.

Formally, we define the procedure of GroundedKG construction from unstructured text to structured representation as $\Phi : \mathcal{T} \to \mathcal{G}$. Given a document represented as a sequence of sentences $S = \{s_1, \ldots, s_N\} \in \mathcal{T}$, we construct a grounded knowledge graph

$$G = \Phi(S) = (V, E),$$

where V denotes the set of nodes and $E \subseteq V \times \mathcal{R} \times V$ denotes the set of directed relational edges.

Each node is represented as a dictionary of attributes:

$$v : \{\text{node\_id}, \text{label}, \text{texts}, \text{node\_type}, \text{grounded\_texts}\} \longrightarrow \text{Values}.$$

Specifically, for each node $v \in V$,

$$\begin{aligned} &v(\text{node\_id}) \in \text{string},\\ &v(\text{label}) \in \text{string},\\ &v(\text{texts}) \in \text{list of strings},\\ &v(\text{node\_type}) \in \{\text{entity}, \text{action}\},\\ &v(\text{grounded\_texts}) \in \text{list of strings}. \end{aligned}$$

Here, *label* is the core concept (e.g., "watch"); *texts* contains textual mentions (e.g., "digital

[2]Extracted entities, relationships, communities, their descriptions, and importance scores are all generated and evaluated by LLMs without verifiable grounding.

watch"); *grounded_texts* contains sentences in the document that the node is grounded to.

Similarly, each edge is represented as a dictionary of attributes:

$$e : \{\text{source\_node}, \text{target\_node}, \text{edge\_role}, \text{edge\_type}, \text{grounded\_texts}\} \longrightarrow \text{Values}.$$

Specifically, for each edge $e \in E$:

$e(\text{source\_node}) \in V$,
$e(\text{target\_node}) \in V$,
$e(\text{edge\_role}) \in \text{String}$,
$e(\text{edge\_type}) \in \{\text{action–entity}, \text{action–action}\}$,
$e(\text{grounded\_texts}) \in \text{List[String]}$.

where *source_node* and *target_node* are the source and target node_ids of the edge; *edge_role* is the semantic role of the relation, e.g. A0; *grounded_texts* contains sentences in the document that the node is grounded to.

3.2. GroundedKG Construction

GroundedKG from SRL parses. Semantic Role Labeling (SRL) is a parsing task that captures the *"who did what to whom, when, where, and how"* structure of a sentence by identifying predicates and assigning semantic roles to their arguments, which describe how each argument participates in an event.

For example, in the sentence *"Peter's mother gave a dose of camomile tea to Peter."* SRL identifies *gave* as the predicate, *Peter's mother* as the agent (A0), *a dose of camomile tea* as the theme (A1), and *to Peter* as the recipient (A2).

We leverage SRL to parse sentences and construct our GroundedKG. For each predicate, we consider its associated components, including core arguments (A0–A3) as well as temporal and spatial arguments when available. We first create the graph nodes: each predicate is represented as an *action node*, while all arguments are represented as *entity nodes*. We then add edges by connecting each predicate to its arguments via *entity–action relations*, for example: `(A1, give, a dose of camomile tea, entity-action, <reference to source text>)`. Finally, for all actions within a sentence, we introduce directed *action–action relations* to encode their temporal order. For the previous example, Peter's mother first makes the tea before giving it to Peter, so the action *give* follows *make*: `(next, make, give, action-action, <reference to source text>)`.

GroundedKG from AMR parses. Abstract Meaning Representation (AMR) is a semantic parsing task that maps a natural language sentence to a rooted, directed acyclic graph encoding its core semantic meaning, abstracting away from syntactic variation. In an AMR graph, nodes correspond to concepts, including actions and entities, and edges represent semantic relations between these concepts.

For example, in the sentence *"Peter's mother gave a dose of camomile tea to Peter."* AMR identifies *gave* as the root action (`give-01` in PropBank), with directed edges to:

- *mother* (A0) as a noun concept which is a person with relations to another person concept *Peter*,
- *tea* as a noun concept as role A1 with quantifier *dose* and modifier *camomile*,
- *Peter* as the person concept (A2).

Similar to SRL graph construction, we consider all the concepts including actions and entities. We first create nodes: each action is represented as an *action node*, while all other associated entities are represented as *entity nodes*, and we add modification words to the node text attribute, for example: `"(tea, [camomile tea, a dose of camomile tea], tea, <reference to source text>)"`. We then link nodes with edges by connecting each action to its associated entities via *entity–action relations*, for example: `"(A1, give-01, tea, entity-action, <reference to source text>)"`. Finally, for all actions within a sentence, we introduce directed *action–action relations* to encode their temporal order, for example: `"(next, make-01, give-01, action-action, <reference to source text>)"`.

3.3. Embedding Techniques

After constructing the GroundedKG, we create our search index by embedding each node into a vector representation that captures both its semantic meaning and its context within the graph structure. We explore three variants of node embedding: *basic node embedding*, *average neighbor embedding*, and *attention-based neighbor embedding*.

Basic Node Embedding. We use a pretrained embedding model f_{embed} to encode the *node name* and *node text* into vector representations. The embedding of a node v is defined as:

$$\text{Embed}(v) = \alpha\, f_{\text{embed}}(v_{\text{name}}) + (1-\alpha)\frac{1}{p}\sum_{i=1}^{p} f_{\text{embed}}(v_{\text{text}}^{(i)}),$$

where v denotes a node in the GroundedKG, $v_{\text{name}} \in \texttt{str}$ is the node name, $v_{\text{text}} =$

$\{v_{\text{text}}^{(1)}, \ldots, v_{\text{text}}^{(p)}\}$ is the set of associated textual descriptions, and $\alpha \in [0, 1]$ is a weighting hyperparameter.

Average Neighbor Embedding. Beyond the basic node embedding that relies solely on a node's name and text, we incorporate the average embeddings of its neighboring nodes to capture local graph context. The resulting node embedding is computed as:

$$\text{NeighborEmbed}(v) = \beta\,\text{Embed}(v) + (1-\beta)\frac{1}{q}\sum_{j=1}^{q}\text{Embed}(v_j),$$

where $\{v_1, \ldots, v_q\}$ are the neighboring nodes of v, and $\beta \in [0, 1]$ is a weighting hyperparameter. This formulation equally weights contextual information from connected nodes, such as the actions performed by an agent or the participants involved in an event.

Attention-Based Neighbor Embedding. To account for varying importance among neighboring nodes, we further introduce an attention-based neighbor embedding. Instead of uniformly averaging neighbor embeddings, we compute attention weights based on cosine similarity between node embeddings. The attention score between node v and its neighbor v_j is defined as:

$$\text{Attn}(v, v_j) = \text{softmax}_j\big(\text{Embed}(v) \cdot \text{Embed}(v_j)\big)$$

where $\cdot$ denotes the dot product, and $v_j \in \{v_1, \ldots, v_q\}$. The final node embedding is then computed as:

$$\text{AttentionEmbed}(v) = \beta\,\text{Embed}(v) + (1-\beta)\sum_{j=1}^{q}\text{Attn}(v, v_j)\,\text{Embed}(v_j).$$

This attention-based neighbor embedding highlights the most relevant contextual information according to semantic similarity, such as salient actions performed by an agent or the primary participants involved in an event.

3.4. Retrieval Techniques

In the querying stage, our goal is to retrieve a subset of query-relevant content $S_{\text{selected}} \subseteq S$ with high recall. Specifically, we aim to select grounded texts that are accurate, non-redundant, and that do not omit relevant evidence required for answering the query. To this end, we design three retrieval techniques to extract and filter relevant content.

For all retrieval techniques, we first parse the user query using the same graph construction procedure Φ defined above. Each query Q is converted into a query graph

$$G_q = \Phi(Q) = (V_q, E_q).$$

Each query node is then embedded following the respective node embedding method described in Section 3.3.

Basic Node-Based Text Retrieval. For each node $u \in V_q$ in the query graph G_q, we compute its cosine similarity with all nodes $v \in V$ in the previously constructed knowledge graph $G_{\text{GroundedKG}}$. We then retrieve the Top-K most similar nodes ($K = 10$ in our experiments) and extract their grounded texts.

Formally, we define the similarity function as

$$\text{sim}(v, u) = \cos\big(\text{Embed}(v) \cdot \text{Embed}(u)\big), \quad v \in V,\ u \in V_q.$$

For each query node $u_i \in V_q$, the Top-K retrieval is defined as

$$\text{TopK}(u_i) = \arg\operatorname*{topK}_{v \in V} \text{sim}(v, u_i).$$

The grounded texts associated with the retrieved nodes are then collected as

$$S_{\text{selected}}(u_i) = \bigcup_{v_{ij} \in \text{TopK}(u_i)} v_{ij}.\text{grounded\_texts}.$$

Finally, the overall selected text set is defined as

$$S_{\text{selected}} = \bigcup_{u_i \in V_q} S_{\text{selected}}(u_i), \quad S_{\text{selected}} \subseteq S.$$

Text Filter with Vector Similarity. In addition to basic node-based retrieval, we further filter the selected texts using vector similarity at the text level, as is commonly done in contemporary RAG systems. Specifically, we compute the cosine similarity between the query representation q and each selected text $s_i \in S_{\text{selected}}$, and retain the Top-$K$ texts with similarity scores higher than a threshold τ:

$$\text{VectorSim}_\tau(q) = \{s_i \in S_{\text{selected}} \mid \text{sim}(q, s_i) \geq \tau\}.$$

Text Filter with Retrieval Count. We additionally introduce a salience-based filtering strategy that measures the importance of each grounded text by counting how frequently it is selected by different retrieved nodes. Intuitively, if a grounded text s is associated with multiple selected nodes, it is more likely to be relevant to the query.

Formally, for a grounded text $s_i \in S_{\text{selected}}$, its salience score is defined as

$$\text{RetCount}(s_i) = \sum_{v \in V_{\text{selected}}} \mathbb{I}\big[s_i \in v.\text{grounded\_texts}\big],$$

where $V_{\text{selected}} = \bigcup_{u \in V_q} \text{TopK}(u)$ and $\mathbb{I}[\cdot]$ is the indicator function.

4. Experiments

4.1. Dataset

NarrativeQA. NarrativeQA (Kočiský et al., 2017) is a large reading comprehension benchmark designed to evaluate deep understanding and reasoning over long texts. It consists of 1,572 full-length stories from books and movie scripts, the split for books are 548 train, 58 validation and 177 test. Annotators wrote 46,765 question–answer pairs (around 30 question-answer pairs for each book) based on human-written summaries rather than on the raw text, which encourages models to perform global, integrative reasoning rather than superficial span extraction. Each query requires synthesizing information distributed throughout entire narratives, necessitating comprehension of characters, events, and their relations across extended context spans. Answers are generally short but often not found as direct spans in the stories, reflecting the task's emphasis on full-content abstraction.

In this experiment, we select three books from the NarrativeQA training split, covering a diverse range of text lengths: The Tale of Peter Rabbit (∼5,000 words), The Phantom of the Opera (∼90,000 words), and Robinson Crusoe (∼120,000 words).

4.2. Metrics

We evaluate model performance using a combination of exact-matching, sequence-level, and semantic similarity metrics, capturing both surface-form accuracy and semantic equivalence between predicted and reference answers.

Exact Match (EM). EM measures whether the predictions exactly contains the reference answers after standard normalization (cleaning up whitespace) and lowercasing. This metric is strict and assigns full credit only when the predicted answer contains an identical segment of the ground truth.

Sequence Match (SM). SM evaluates whether the predicted answer contains the reference sequence at the token level. Unlike Exact Match, it assigns full credit when predicted answer (e.g. his shoes and his jacket) contains correct reference token sequences (e.g. his shoes and jacket), thereby providing a softer measure of correctness for answers that are lexically similar but not identical.

ROUGE-L. ROUGE-L (Lin, 2004) measures the longest common subsequence (LCS) between the predicted and reference answers. It captures both precision and recall of overlapping subsequences without requiring contiguous matches, making it effective for evaluating content preservation in free-form or abstractive generation tasks.

BERTScore. BERTScore (Zhang et al., 2020) computes semantic similarity between predicted and reference answers by aligning contextualized token embeddings from a pretrained transformer model (e.g., BERT). Unlike lexical overlap metrics, BERTScore accounts for paraphrasing and semantic equivalence, providing a robust measure of meaning-level similarity.

4.3. Experimental Settings

Optimal Settings. The best-performing variant of GroundedKG-RAG uses graph construction based on AMR parses, basic node embedding, $K = 10$ most similar nodes, and no additional text filters from vector similarity or retrieval counts.

Text Segmentation. Documents are segmented using the LangChain recursive character text splitter into chunks of 5,000 tokens with zero overlap, following paragraph boundaries. Each chunk is further segmented into sentences using spaCy with the `en_core_web_sm` model.

Coreference Resolution. Coreference resolution is applied to each sentence using spaCy for named entity detection and fastcoref for coreference detection. Entities are grouped and grounded to the original sentence. Each sentence is stored as a tuple `(sent_index, original_sentence, normalized_sentence, coref_modified_sentence)` after processing.

Sentence Parsing. Semantic parsing is performed on each coreference-resolved sentence using either a semantic role labeling (SRL) parser or an abstract meaning representation (AMR) parser. For SRL parsing, we use spaCy for verb predicate identification and `cu-kairos/propbank_srl_seq2seq_t5_large` for argument extraction. For AMR parsing, we use amrlib for AMR graph extraction together with Penman for graph parsing.

Graph Construction. We employ networkx library for GroundedKG construction and pyvis for storing the graph in JSON and html formats.

Embeddings. Nodes in the constructed graph are embedded using the lightweight sentence embedding model `all-MiniLM-L6-v2`. For stability, the vector is normalized after each step, including node embedding, aggregated neighbor embeddings, combined node neighbor embeddings.

Retrieval-Augmented Generation. Queries are processed using the same pipeline as documents. Relevant sentences are retrieved based on

graph matching and provided as context to OpenAI models `gpt-5-2025-08-07` and `gpt-5-nano-2025-08-07` for question answering. We did not observe a big difference in the two model's answer qualities, and therefore selected gpt-5-nano for our further experiments.

Hardware. All experiments are conducted using a single NVIDIA T4 or L4 GPU accessed on Google Colab, and the OpenAI API with batch processing is used for answer generation.

5. Results

We evaluate our GroundedKG-RAG by comparing with different baselines and ablating our design choices.

Baseline Comparison. Our main results are shown in Table 1. We compare our GroundedKG with a closed-book LLM (without any context, *no context*), an open-book LLM (offering the full book content as context, *full context*), and GraphRAG. Since the NarrativeQA questions are based on copyright-free books from Project Gutenberg, we assume that both the books and potentially also the NarrativeQA question-answer pairs have been part of the proprietary LLM's training data. The *no context* baseline establishes the performance of the answer generation model when only retrieving (book text or memorized NarrativeQA answers) from the model weights.

Across the three books, our performance is much higher than the *no context* baseline. For *full context*, we achieve three percent improvement in both BERTScore and Rouge F1 on Peter Rabbit, from 59 to 62 and 32 to 34. For Phantom of the Opera and Robinson Crusoe, GroundedKG-RAG's Rouge-L F1 is relatively comparable to the full content with only 1/3 of its total length. The BERTScore drops from 62 to 56 and 55 to 46 compared to the full context, showing that longer books are still more challenging for the much bigger knowledge graph we built for these books. Compared to GraphRAG we achieve similar performance on BERTScore, 62 compared to 63, but better performance on ROUGE-L, 34 compare to 8, indicating the difference in groundedness of the answers. Due to the high number of LLM calls performed by GraphRAG, we limit our evaluation of the model to experiments on Peter Rabbit.

Sentence Parsing. Comparing knowledge graph construction from SRL parses with AMR parses, we see in Table 2 that AMR parses improve on SRL parses in both BERTScore and ROUGE-L (52 vs. 58 and 30 vs. 34). This may be the result of the finer granularity level allowing more accurate node matches between query nodes and GroundedKG nodes. For example, in SRL parsing nodes, every token contributes equally to the node embedding, thus *Peter's father* may have a higher cosine similarity score with *Peter's mother* than *the four little kids' father*. In AMR parsing, the core tokens determine the node embedding, thus the embedding of *father* for both *Peter's father* and *the four little kid's father* will lead to a match.

TopK. We test the choice of TopK. $K = 5$ receives lower BERTScore than $K = 10$. The ROUGE-L is the same in Table 2. Although the true grounded sentence has a higher probability in the first several nodes, we assume selecting more nodes allows more grounded sentences to be incorporated in the context, leading to better performance in the rare case that the gold answer is a reflection of multiple sentences. For instance, the node *Peter* can be matched to a similar node *Peter's jacket*, which is not in the very top of the matched nodes when compared to *Peter*, *Peter's sister* etc., yet contains the correct answer to the question.

Embedding. We ablate the performance of our three embedding techniques on Peter Rabbit: basic node embedding, average neighbor embedding, and attention-based neighbor embedding. The three settings achieve similar scores on BERTScore. The original node embedding achieves the best score in BERTScore with 62 compared to 60 for others. The attention-based neighbor embedding gets the best score in ROUGE-L with 36. The average neighbor embedding gets the lowest scores. This is because the outliers in an average of embeddings can have a big influence on the node embedding, modifying up to 50% of the top-10 retrieval results. The attention-based neighbor embedding is more stable compared to the average neighbor embedding and only changes 10-20% of the retrieved results (often in the last ranks). Since, the contextual embedding, which considers the graph structure and the neighbors of a node does not exhibit an obvious gain on the task, we use the basic node embedding as our primary setting.

Retrieval. We consider three different settings in the retrieval stage: basic node grounded_text retrieval, filtering with sentence vector cosine similarity, and filtering with sentence retrieval counts. For sentence cosine similarity, the peformance drops 4 percent on BERTScore and 1 percent on ROUGE-L compared with grounded_text retrieval for GroundedKG-RAG, and 1 percent in BERTScore and 5 percent in ROUGE-L when using SRL parses and $K = 5$ on Peter Rabbit (see Table 2). For filtering based on sentence retrieval counts, the BERTScore drops from 46 to 43 and F1 drops from 26 to 23 on Robinson Crusoe (see Table 4 in the appendix). We conjecture that filtering can remove the gold grounded sentences for some of the questions, leading to worse question

Model	Exact match	Sequence match	Bertscore	RougeL F1
Peter Rabbit: 300 nodes, 700 edges				
No context	**16**	16	49	22
Full context	13	16	59	32
GraphRAG	**16**	20	**63**	8
GroundedKG-RAG	**16**	**23**	62	**34**
The Phantom of the Opera: 25k nodes, 56k edges				
No context	17	17	49	25
Full context	**34**	**37**	**62**	**36**
GroundedKG-RAG	**34**	**37**	56	**36**
Robinson Crusoe: 33k nodes, 80k edges				
No context	13	13	25	8
Full context	36	36	**55**	**31**
GroundedKG-RAG	**40**	**43**	46	30

Table 1: Main result of GroundedKGRAG compared to baselines on our three selected books.

Model	Exact match	Sequence match	Bertscore	RougeL F1
No context	**16**	16	49	22
Full context	13	16	59	32
GroundedKG-RAG with SRL and $K = 5$	10	16	52	30
GroundedKG-RAG with SRL, $K = 5$, and VectorSim ($\tau = 0.2$)	10	13	51	25
GroundedKG-RAG	**16**	**23**	**62**	34
GroundedKG-RAG with $K = 5$	13	20	58	34
GroundedKG-RAG with VectorSim ($\tau = 0.2$)	13	16	58	33
GroundedKG-RAG (basic Embed)	**16**	**23**	**62**	34
NeighborEmbed ($\beta = 0.8$)	**16**	**23**	60	33
AttentionEmbed ($\beta = 0.5$)	13	20	60	**36**
AttentionEmbed ($\beta = 0.8$)	**16**	**23**	60	**36**

Table 2: Ablation of our GroundedKG-RAG on Peter Rabbit.

answering performance. During error analysis, we find that for the question *"What did Peter do after arriving home?"*, the sentence *"Peter arrives home exhaustedly."* will have a high similarity score, while the true grounded text *"Peter gets some food and runs to bed because he is too exhausted."* receives a similarity score below the threshold because of the smaller number of matching tokens between the sentence and the question. The same happens for the retrieval count filter.

6. Error Analysis

We distinguish between (1) KG construction, (2) retrieval and (3) answer generation errors. The discussion of mapping *"the four little kids' father"* to the father concept is an example of an error of the first kind (see Section 5). This works better in graphs built from AMR parses than from SRL parses. The second kind is a failure to retrieve relevant nodes from a given query (see an example in Table 5 in the appendix). The third kind of error occurs when the answer generation model gets distracted by seemingly relevant information in the context, cannot handle the length of the context, or ignores the context and therefore the instruction (see third error type in Table 5).

7. Graph Case Study

We present a simple example to illustrate the creation of our GroundedKG from SRL and AMR parses in Table 3. We can clearly observe that AMR-based graph grounded coreference-resolved entities ("it" has been resolved to "some camomile tea" in the second clause) into the same node, in this case *tea*, while SRL created a duplicate node for it. The same happens with the two mentions of Peter ("Peter" and "to Peter"). The AMR graph avoids duplicate node creation and improves node matching during retrieval.

Input Text: Peter's mother put Peter to bed and made some camomile tea; and Peter's mother gave a dose of some camomile tea to Peter!

AMR Graph:

```
(a / and
    :op1 (p / put-01
        :ARG0 (p2 / person
            :ARG0-of (h / have-rel-role-91
                :ARG1 (p3 / person
                    :name (n / name
                        :op1 "Peter"))
                :ARG2 (m / mother)))
        :ARG1 p3
        :ARG2 (b / bed))
    :op2 (m2 / make-01
        :ARG0 p2
        :ARG1 (t / tea
            :mod (c / camomile)
            :quant (s / some)))
    :op3 (d / dose-01
        :ARG0 p2
        :ARG1 p3
        :ARG2 t))
```

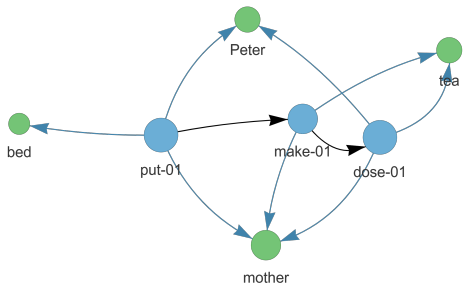
SRL Graph:

```
Predicate: put
    ARG-0: Peter's mother
    ARG-1: Peter
    ARG-2: to bed

Predicate: made
    ARG-0: Peter's mother
    ARG-1: some camomile tea

Predicate: gave
    ARG-0: Peter's mother
    ARG-1: a dose of some camomile tea
    ARG-2: to Peter
```

GroundedKG with AMR:



GroundedKG with SRL:

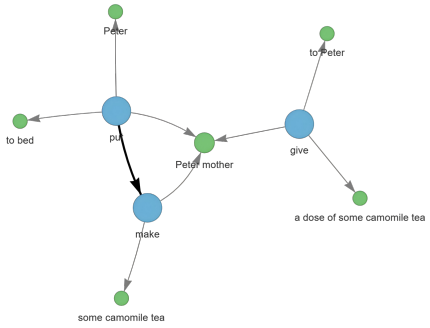


Table 3: Example comparing AMR and SRL graphs.

8. Conclusion

In this work, we propose the GroundedKG-RAG framework, where each node and edge in the GroundedKG is directly extracted from and grounded in the source document at the sentence level. The nodes represent the entities and actions, and the edges represent the temporal or semantic relation between them. For the GroundedKG construction, we experimented with two different document parsing methods: SRL and AMR. Due to better semantic concept identification in AMR, it achieves higher scores on NarrativeQA than SRL. In our GroundedKG, we embed the nodes to incorporate the contexts. The graph's edges serve to relate nodes' temporal and semantic relationship. We evaluate three types of embedding techniques, and find that including neighboring nodes' information in a node's embedding does not significantly improve retrieval results. We also experiment with filtering retrieved grounded texts through either cosine similarity with the query embedding or retrieval counts, but neither filtering improves evaluation scores. Error analysis shows that these filters remove the gold sentence in a few evaluation examples.

In future work, we plan to further improve our GroundedKG-RAG in the following directions: (1) there is still room for improvement in the node matching, so different approaches to the graph structure, node/edge composition, used relations, and embedding techniques can be tried. (2) Even though we were not successful with our experiments in filtering out irrelevant source sentences, we believe that the answer generation model would profit from a less fragmented and sometimes contradicting context, giving an opportunity for better filtering techniques. (3) GroundedKG-RAG's scores improving over the *full context* baseline for the short Tale of Peter Rabbit, but BertScore is lagging behind on longer books shows that the large knowledge graphs constructed for these books may benefit from better graph retrieval techniques.

9. Acknowledgements

We thank getAbstract for the support and partial funding of this project.

10. Bibliographical References

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2025. From local to global: A graph rag approach to query-focused summarization.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2025. LightRAG: Simple and fast retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 10746–10761, Suzhou, China. Association for Computational Linguistics.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: neurobiologically inspired long-term memory for large language models. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. From RAG to memory: Non-parametric continual learning for large language models. In *Forty-second International Conference on Machine Learning*.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2017. The narrativeqa reading comprehension challenge.

Zhuowan Li, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. 2024. Retrieval augmented generation or long-context LLMs? a comprehensive study and hybrid approach. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 881–893, Miami, Florida, US. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations (ICLR)*.

V.A. Traag, L. Waltman, and N.J. van Eck. 2019. From louvain to leiden: guaranteeing well-connected communities. In *Sci Rep 9, 5233*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert.

A. Question Answering Prompt

We used the following prompt format to answer questions with and without context.

With content: `Please answer the question based on the following content:` `Content:` $< content >$ `Question:` $< question >$ `Answer:`

Without content: `Please answer the question based on your memory of the book` $< book\_name >$`:` `Question:` $< question >$ `Answer:`

B. Sentence Count Ablation

A small ablation on Robinson Crusoe for the text filter with retrieval count is shown in Table 4.

C. Error Analysis

An error analysis with examples for the three types of errors we encounter is shown in Table 5.

Model	Exact match	Sequence match	Bertscore	RougeL F1
	Robinson Crusoe			
No context	13	13	25	8
Full context	36	36	**55**	**31**
GROUNDEDKG-RAG	**40**	**43**	46	30
GROUNDEDKG-RAG with RetCount ≥ 2	30	30	43	23

Table 4: Comparing AMR with sentence count of 2 or more on Robinson Crusoe.

Tyep I Error	
Question	What happened to the young rabbits' father?
Matched Node	to the young rabbits' father ['with four little rabbits mother', 'The Tale Of Peter Rabbit', 'from Peter cousin, little Benjamin Bunny'] happen ['happen_112', 'come_100', 'come_113']
Grounded Text	text_0-3, text_0-6, text_0-20, text_0-21, text_0-27, text_0-37, text_0-40, text_0-42
Predicted Answer	The text does not mention the father of the young rabbits.
Gold Answer	Mrs. McGregor baked him into a pie.
Solution	Improve GROUNDEDKG construction for node matching
Type II Error	
Question	What does Peter have for dinner after getting back home?
Matched Node	Peter ['Peter', 'Peter back', 'First Peter'] for dinner: [for supper, eat_26, eat_27] have: [have_142, not have_66, get_131] Peter: [Peter, Peter back, First Peter] back home: [home, away, go back_109] get: [get_131, not get_18, give_2]
Grounded Text	text_0-1, text_0-8, text_0-13, text_0-14, text_0-15, text_0-16, text_0-17, text_0-18, text_0-20, text_0-21, text_0-23, text_0-24, text_0-25, text_0-27, text_0-28, text_0-29, text_0-30, text_0-31, text_0-33, text_0-35, text_0-36, text_0-37, text_0-39, text_0-40, text_0-41, text_0-42, text_0-43, text_0-44, text_1-0, text_1-1, text_1-2, text_1-4, text_1-5, text_1-7, text_1-9, text_1-11 text_1-9 His mother put him to bed and made some camomile tea;; and she gave a dose of some camomile tea to Peter! text_1-11 But– Flopsy, Mopsy and Cottontail had bread and milk and blackberries for supper.
Predicted Answer	Bread and milk and blackberries.
Gold Answer	Chamomile tea
Solution	Filter out distracting results and use a stronger question answering model to distinguish the correct answer from misleading ones.
Type III Error	
Question	Where does Peter see his lost clothing?
Matched Node	Peter ['Peter', 'Peter back', 'First Peter'] see ['see_116', 'look_30', 'look_71'] clothing ['his jacket', 'the little jacket and the shoes', 'underneath'] lose ['lose_41', 'lose_42', 'lose_136']
Grounded Text	text_0-13, text_0-14, text_0-15, text_0-16, text_0-17, text_0-18, text_0-19, text_0-20, text_0-21, text_0-23, text_0-24, text_0-25, text_0-26, text_0-27, text_0-28, text_0-29, text_0-30, text_0-31, text_0-33, text_0-35, text_0-36, text_0-37, text_0-39, text_0-40, text_0-41, text_0-42, text_0-43, text_0-44, text_1-0, text_1-1, text_1-2, text_1-3, text_1-4, text_1-5, text_1-7, text_1-9 text_1-3 Mr. McGregor hung up the little jacket and the shoes for a scare-crow to frighten the blackbirds.
Predicted Answer	In the tool-shed.
Gold Answer	On McGregor's scarecrow
Solution	Better filtering and using a question answering model that can reason over long and similar contexts.

Table 5: Error analysis with an example for each type of error.