Retrieval-Augmented Generation with Knowledge Graphs: A Survey

***Abstract*—Retrieval-Augmented Generation (RAG) has recently emerged as a powerful framework to enhance Large Language Models (LLMs) by leveraging external knowledge sources. However, traditional RAG systems relying on flat data structures often struggle with complex relationships and semantic reasoning. This survey explores the integration of knowledge graphs (KGs) into RAG systems, which offers structured information storage, enhanced semantic understanding, and dynamic update capabilities. We systematically review current methodologies across the stages of graph construction, graph retrieval, and augmented generation, highlighting key innovations and challenges. Additionally, we examine datasets, evaluation metrics, and experimental frameworks, and propose future research directions to advance RAG systems. This paper serves as a comprehensive resource for understanding the state-of-the-art and identifying open problems in this emerging field.**

***Index Terms*—Retrieval-Augmented Generation, Large Language Models, Knowledge Graphs**

I. Introduction

WHILE Large Language Models (LLMs) have demonstrated remarkable capabilities in understanding and generating human language, they also have some limitations. When LLMs encounter queries for which they do not have a definitive answer based on their training data, they may generate plausible but inaccurate or fabricated information. This phenomenon is referred to as "hallucination" [1]. Meanwhile, LLMs cannot always grasp the latest information and domain-specific knowledge. The introduction of Retrieval-Augmented Generation (RAG) [2] partially mitigates these issues by combining retrieval and generation, allowing LLMs to leverage external knowledge bases to enhance their response capabilities. The fundamental principle of RAG is to retrieve the most relevant text chunks in external knowledge bases according to the query, and then input them into LLMs together with the query as prompts, which leads to an appropriate generation. RAG technology improves the timeliness and accuracy of LLMs' responses, and enables LLMs to acquire additional knowledge without retraining, thus making it receive widespread attention.

Despite the impressive performance and convenience of RAG, it still faces several challenges. First of all, external documents may contain information that is not completely relevant to the query. Simply retrieving relevant documents and adding all of them to the prompts will introduce noise, resulting in decreased accuracy of LLMS [3, 4]. Noisy information may also make the retrieved documents too long to exceed the limits of LLM context windows. In addition, most external data in traditional RAG are stored as flat data representations, restricting the ability of LLMs to understand the complex relationships between entities. Similarly, RAG cannot summarize global information between documents as well, and hence struggles with Query-Focused Summarization (QFS) tasks [5]. Finally, incremental update to large text databases is also a challenge.

Knowledge Graphs (KGs) have been considered in recent studies to address the above challenges. In these studies, unlike flat data representations in traditional RAG, external data is summarized into a knowledge graph structure by applying some knowledge extraction techniques. Then, in the retrieval stage, specific graph structures instead of text chunks are retrieved based on the query. After that, retrieved graph structures will be converted into text formats which LLMs can accept and input into LLMs together with the query as prompts to generate an answer. Knowledge graphs show the following advantages in this process: (1) *Structured information storage*: The knowledge graph stores external textual information in the form of a graph, making the data more structured, removing unnecessary noise, and reducing the amount of text input into the LLMs context window. (2) *Enhanced semantic understanding*: By leveraging entities and relations in knowledge graphs, LLMs can gain a deeper understanding of the semantics of queries and improve the relevance and accuracy of generated answers. LLMs can also better perform tasks such as multi-hop reasoning and QFS based on the graph structure. (3) *Support for dynamic updates*: Knowledge graphs can usually be updated dynamically, allowing RAG systems to obtain the latest information in real time and ensure that the generated content reflects the latest knowledge.

This paper aims to investigate the application technology of knowledge graphs in RAG systems, build a complete workflow architecture, summarize and classify existing methods according to different principles in each stages of RAG. Our contributions are as follows:

- We summarize and divide the general workflow of RAG with knowledge graph based on the current state-of-the-art methods.
- We investigated the principles of current methods and organized them into categories in stages of graph construction, graph retrieval and augmented generation.
- We discuss related experimental benchmarks and metrics, and provide prospects for future research directions.

II. Background

In this section, we introduce relevant background knowledge to better understand the following content.

A. Large Language Models

Large Language Models (LLMs) represent a significant advancement in the field of natural language processing (NLP). These models, which are trained on vast amounts of text data,

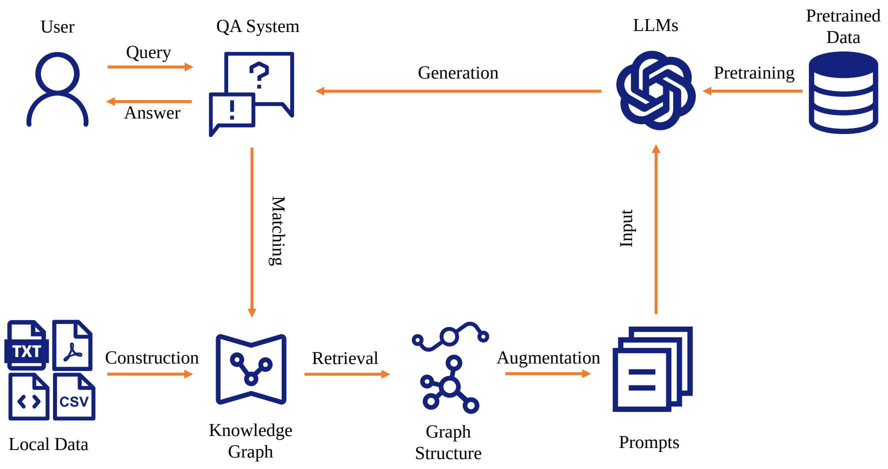

Fig. 1: Overview of knowledge graph RAG systems.

leverage deep learning techniques to understand and generate human-like language. By capturing intricate patterns and relationships within the data, LLMs can perform a wide range of tasks, including text completion, translation, summarization, and question-answering. Their ability to generate coherent and contextually relevant responses has opened new avenues for applications in various domains, from automated customer service to creative writing, thereby transforming the landscape of human-computer interaction.

B. Knowledge Graphs

Knowledge Graphs (KGs) are structured representations of data, usually denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \{\mathbf{x}_v\}_{v\in\mathcal{V}}, \{\mathbf{e}_{i,j}\}_{i,j\in\mathcal{E}})$, where $\mathcal{V}$ is a set of nodes representing concept entities, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges representing the relationships between these entities [6]. Additionally, each node or edge may posses a set of textual attributes, represented as $\{\mathbf{x}_v\}_{v\in\mathcal{V}}$ and $\{\mathbf{e}_{i,j}\}_{i,j\in\mathcal{E}}$, respectively.

Nodes/edges and attributes can also be organized into key-value pairs, where the key is the name of the entity/relationship and the value is the corresponding textual description. This can make it more convenient for retrieval and generation stage of RAG [7].

C. Graph Neural Networks

Graph Neural Networks (GNNs) have become a fundamental tool in handling graph-structured data. GNNs mainly consist of two parts: message passing and aggregation, which obtain node and graph representations by aggregating graph structure information:

$$\mathbf{h}_i^{(l)} = \mathbf{AGG}_{j\in\mathcal{N}(i)}(\mathbf{h}_i^{(l-1)}, \mathbf{MSG}(\mathbf{h}_j^{(l-1)}, \mathbf{e}_{i,j}^{(l-1)})) \tag{1}$$

where $\mathbf{h}_i^{(l)}$ denotes node representations in the l-th layer, $\mathcal{N}(i)$ represents the neighbors of node i, and $\mathbf{e}_{i,j}^{(l)}$ denotes information of edge between node i and node j. $\mathbf{MSG}$ and $\mathbf{AGG}$ represent the message passing function and aggregation function, respectively, and their implementations can vary significantly across different GNNs. For instance, GCN [8] utilizes convolutional neural networks to aggregate information from neighboring nodes, while GAT [9] employs attention mechanisms for node information aggregation. Finally, graph representations can be obtained by pooling the node representations [10].

In knowledge graph RAG systems, GNNs can be utilized to model graph structure retrieved, which can well combine the semantic and structural information of the knowledge graph in the augmented generation stage.

III. Architecture

In this section, we introduce the overall architecture of RAG with knowledge graph. We show the architecture overview of it in Figure 1. We divide this architecture into three essential stages: graph construction, graph retrieval and augmented generation. In graph construction stage, various types of local files are constructed into a knowledge graph through specific methods. Then, in the retrieval stage, the user query and the knowledge graph are matched according to the semantic relevance, and the most relevant graph structures will be selected, which can be in the form of different granularity. Finally, in the augmented generation stage, the retrieved graph structures are first converted into data formats acceptable to LLMs, and then input into LLMs together with the query as prompts to generate the answer.

In the rest of this section, we will discuss the specific details of each stage, categorize and summarize the various methods that have been employed. The categorization is shown in Table I.

A. Graph Construction

The quality of the knowledge graph directly influences the performance of RAG. Therefore, how to construct a dynamic knowledge graph from multi-source data and maintain its timeliness and accuracy is the key to this stage.

1) Graph Categories: As described in Section 2, the traditional entity-relation knowledge graph consists of a set of nodes and edges, where nodes represent entities and edges

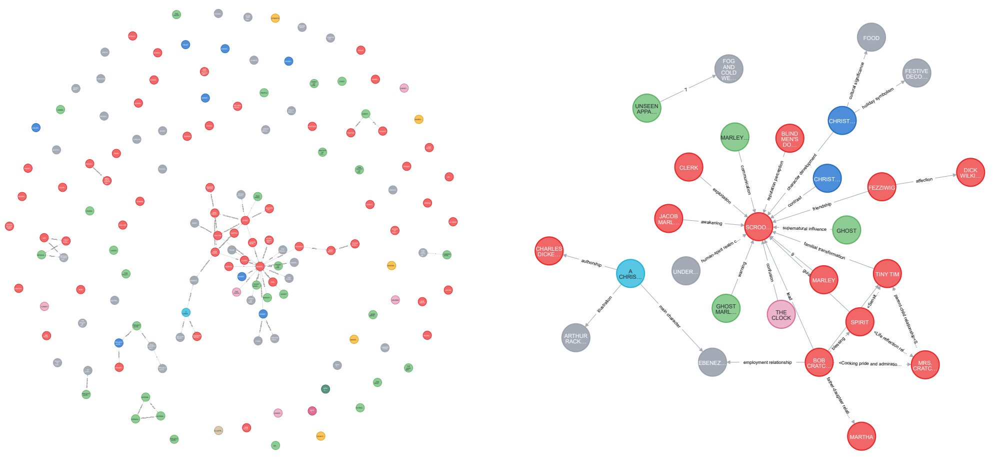

Fig. 2: An example of entity-relation knowledge graph, which is constructed from the novel *A Christmas Carol*. The left picture shows the complete knowledge graph, which includes all entities and relationships in the novel. The right picture shows the central part of the knowledge graph, which shows the interpersonal relationships of the protagonist Scrooge. We utilize the *qwen2* [11] language model and the *nomic-embed-text* [12] embedding model to assist the construction of this graph.

represent the relationships between entities. Knowledge extraction is applied to obtain usable knowledge units, including entities, relations, and attributes from natural language text input. There are many automated or semi-automated extraction technologies available. Hearst et al. [13] utilize rule-based methods to manually extract knowledge from text, most of which are formulated by domain experts. Brin et al. [14] exploit statistical features and machine learning algorithms to automatically learn knowledge from data. More common approaches is to utilize pre-trained language models (such as BERT, GPT) for knowledge extraction [5, 7, 15]. We give an example of an entity-relationship knowledge graph in LightRAG [7], which utilizes LLMs to extract all entities and relationships in novel *A Christmas Carol*. We visualizes them through the neo4j database in Figure 2.

Knowledge extraction can't be completly precise, leading to proposals for other kind of knowledge graphs. Typical one of them is document-structure graph. Wang et al. [16] directly utilize text passages as nodes and builds edges based on whether there is semantic similarity between passages. They use co-occurring words, paragraph embedding matching, and common Wikipedia entities to determine whether there is semantic similarity between passages. Li et al. [17] utilize structural information and shared keywords to construct the graph of passages. Munikoti et al. [18] regard documents as nodes and construct edges based on the co-citation, co-topic, co-location, and co-institution relationships between documents. These methods can quickly build a knowledge graph without meeting extraction errors.

2) Index Storage: After construction of knowledge graph, graph structures should be retrieved according to the index. Diverse index types have been employed in current methods. Some directly utilize the graph as index and employ traditional graph traversal algorithms for retrieval [16, 19].

Other methods convert graph data into vector representations and store them in vector database for fast retrieval. For example, He et al. [20] encode textual information of each node and edge using LLMs while Li et al. [21] encode triples into text embeddings. Hu et al. [22] embed each k-hop ego graph into indexes. Edge et al. [5] utilize community detection algorithms to partition the knowledge graph into different communities, and LLMs are employed to generate textual summaries for each community, which are then embedded into a vector space. Guo et al. [7] organize nodes/edges and their text attributes into key-value pairs and store the keys in vector space.

3) Knowledge Update: In rapidly changing data environments, it's crucial to update the knowledge database efficiently. Current methods are divided into non-incremental update and incremental update. Non-incremental update means that at each time new data is added, the knowledge graph or index needs to be rebuilt, while incremental update only requires minor adjustments. Methods that directly ultilize the graph as an index or retrieve nodes/edges generally support incremental update, while methods that retrieve subgraphs usually are non-incremental update. For instance, GraphRAG [5] generates community summaries for each graph community and index them, when the knowledge graph is updated, the community needs to be re-divided and the above process needs to be repeated. GRAG [22] encodes all k-hop ego graphs into indexes, and updates require re-searching all k-hop ego self-graphs. LightRAG [7] support incremental update for it only requires adding new nodes and edges to the knowledge graph and embedding them into the vector database.

B. Graph Retrieval

Graph retrieval refers to the process of finding the most semantically relevant graph structures from the knowledge graph based on the user query. Next, we will categorize

TABLE I: Summary of Existing Methods

Method	Knowledge Graph Type	Index Type	Retrieval Granularity	Retrieval Process
GraphRAG [5]	entity-relation	vector	subgraphs	multi-stage
LightRAG [7]	entity-relation	vector	subgraphs	multi-stage
KG-RAG [15]	entity-relation	graph	paths	iterative
KGP [16]	document-structure	graph	nodes	iterative
GNN-Ret [17]	document-structure	vector	nodes	iterative
ATLANTIC [18]	document-structure	vector	nodes	iterative
GraphCoT [19]	entity-relation	graph	paths	iterative
G-Retriever [20]	entity-relation	vector	subgraphs	multi-stage
Li et al. [21]	entity-relation	vector	triplets	one-time
GRAG [22]	entity-relation	vector	subgraphs	multi-stage
GNN-RAG [23]	entity-relation	vector	triplets	iterative
HippoRAG [24]	entity-relation	vector	nodes	one-time
Pullnet [25]	entity-relation	graph	nodes	iterative
CPR [26]	entity-relation	vector	paths	multi-stage
ToG [27]	entity-relation	graph	paths	iterative
KG-GPT [28]	entity-relation	graph	nodes	multi-stage

and summarize the retrieval granularity, retrieval level, and retrieval process of various methods.

1) Retrieval Granularity: Retrieval granularity refers to the smallest unit of the graph structure retrieved in response to a query. It is influenced by the type of indexing employed and the retrieval algorithm utilized. Each level of retrieval granularity possesses distinct characteristics and is suited to specific application scenarios. We divide the granularity into nodes/edges, triplets, graph paths and subgraphs.

Retrieving nodes and edges is primarily for obtaining detailed information. For instance, Sun et al. [25] retrieved entities from a knowledge graph constructed from an open-domain dataset to solve simple problems in open-domain QA. Specifically, for the document-structure graph mentioned earlier, the retrieval granularity is generally a node because different passages rarely form a meaningful overall structure. For example, Wang et al. [16], Li et al. [17] and Munikoti et al. [18] retrieve passage nodes in the document-structure graph as text prompts directly.

The retrieval of triplets and paths granularity can capture distant relationships between different entities in the query, enhancing contextual understanding and reasoning capabilities, which is often used in reasoning tasks. Li et al. [21] utilize a specific template to convert triples into texts, and then stored them into vector indexing for retrieval. Retrieving paths is an NP-hard problem, because the number of possible paths grows exponentially as the size of the graph increases. One solution is to utilize depth-limited search such as CPR [26] and GNN-RAG [23], and another solution is to use beam search like ToG [27].

The retrieval of subgraphs granularity is the most commonly utilized due to its ability to offer comprehensive understanding to the contextual information. Like paths, retrieving subgraphs is also an NP-hard problem. Therefore, many methods seek to approximatly optimal subgraphs instead of optimal subgraphs. Guo et al. [7] construct one-hop ego graph around retrieved nodes. Hu et al. [22] directly retrieve k-hop ego graph and then implement soft-pruning to remove irrelevant nodes and edges. He et al. [20] design an approximate Prize-Collecting Steiner Tree (PCST) algorithm to construct subgraphs with retrieved nodes and edges. Edge et al. [5] retrieve community summary of relevant subgraphs.

2) Retrieval Level: Retrieval level refers to which level of the knowledge graph the retrieval focuses on. It can be categorized into local-level retrieval and global-level retrieval, which is generally related to the retrieval granularity and downstream tasks.

Local-level retrieval focuses on extracting fine-grained information from the graph. It aims to identify specific nodes, edges, or attributes that are directly relevant to the query. This approach is particularly useful in scenarios where precise and rich data is required, such as in knowledge-based question answering or when generating highly specific content. Global-level retrieval, on the other hand, operates at a higher abstraction level, focusing on broader concepts or themes represented in the graph. This level seeks to capture the overarching context or narrative associated with a set of related entities or events. This method is beneficial for generating summaries or overviews, like QFS tasks. Some methods support both levels of retrieval, which retrieve both local details and global thematic information. For instance, LightRAG [7] first extracts the local and global keywords of the query, then utilize the local keywords to retrieve entities and the global keywords to retrieve relationships, respectively, forming a dual-level retrieval paradigm.

3) Retrieval Process: After query matching, some methods directly utilize the retrieved content as retrieval results, while some methods process these contents to generate better retrieval results, and some methods utilize these contents to perform new retrieval on the graph. According to these retrieval processes, we divide current methods into one-time retrieval, multi-stage retrieval, and iterative retrieval.

One-time retrieval involves comparing the knowledge graph content with the query just once, and then retrieving all graph structures as results. This process is highly efficient and preserves the effective information of the graph, but it may also contain a lot of irrelevant redundancy. They usually embed the retrieval object into the vector space in advance, and then directly retrieve the most relevant information based on the embedding similarity between the query and the retrieval object. For instance, Gutierrez et al. [24] extract node information directly from the graph database and Li et al. [21]

directly extract triplets information from a pre-constructed text index.

Multi-stage retrieval refers to adjusting the structure or content of the retrieved information rather than directly using it as the retrieval result. GraphRAG [5] utilizes LLMs to score the retrieved community summaries and sort them in descending order, removing those with lower scores. GRAG [22] performs soft-pruning on all retrieved k-hop subgraphs, applying deep learning methods to mask irrelevant nodes and edges in the subgraphs. G-Retriever [20] retrieves the nodes and edges most relevant to the query and assign weights to them, and then designs an approximate PCST optimization algorithm to extract the most relevant subgraph. LightRAG [7] gathers neighboring nodes within the local subgraphs of the retrieved graph elements to enhance the query with higher-order relatedness.

Iterative retrieval refers to re-searching the information depending on the results of prior retrievals to improve relevance. Differences between iterative retrieval and multi-stage retrieval may be that the query is reused by subsequent processes in iterative retrieval. For example, KGP [16] first extracts the passage nodes most relevant to the query, then feeds the passage words and the query into LLMs to generate a new description, based on which it retrieves the next possible passage node. ToG [27] utilizes LLMs to select possible reasoning paths, and then feeds them and the query into the same LLM for further selecting until the model believes that current path can infer the answer to the query. GNN-RAG [23] first obtains the subgraph containing the query entity through dense retrieval, then utilizes GNNs to label possible answer entities according to query relevance, and finally retrieves paths between the query entity and the answer entities.

C. Augmented Generation

After retrieving the graph structures related to the query, the last step is to convert them into proper input formats acceptable to LLMs and then input them into LLMs together with the query as prompts to generate the answer. In this process, there are various methods to convert graph structures into textual formats, and many augmented generation techniques have been proposed to improve the quality of the output.

1) Text Conversion: Graph structures such as nodes/edges, triples, paths and subgraphs obtained in the retrieval stage cannot be directly input into LLMs, so they need to be converted into texts while retaining the structural information of the graph. The most direct way is to express the graph structure in natural language, using serial numbers plus texts to express the relative positions of nodes and edges, such as G-Retriever [20]. For methods with retrieval granularity of triples or paths, a common conversion method is to utilize node sequences [23, 27], where reasoning paths are verbalized as "`{entity}` $\rightarrow$ `{relation}` $\rightarrow$ `{entity}` $\cdots \rightarrow \cdots$ `{entity} \n`". It is also very efficient to directly add the text attributes corresponding to the retrieved graph structure to prompts [7, 16].

It is difficult to determine whether LLMs really understand the topology information in above methods. Another approaches include the topological structure of the graph in the text through summarization. Edge et al. [5] generate summaries of subgraphs by LLMs and then integrate results of different community summaries. Hu et al. [22] introduce a novel prompting method based on graph and tree traversals to convert textual subgraphs into hierarchical text descriptions without losing both textual and topological information.

2) Augmentation Methods: During the generation stage, in addition to converting the graph structures into text formats, the contents could also be optimized and enhanced to improve the quality of the output. There are several reasons for this procedure. For instance, the volume of information retrieved may exceed the LLM's context window, or lengthy contexts may make it difficult for the LLM to capture distant entity relationships. Additionally, the graph may contain low-relevance content, which can hinder the LLM's ability to focus on key information. Edge et al. [5] generate summaries for each graph community and generate answers for each summary based on the query during the retrieval process. Then they utilize LLMs to score each community answer to reflect whether it can well answer the query, and add community answers to prompts in descending order of score until the context window limit of LLMs is reached. Hu et al. [22] apply deep learning algorithm to mask retrieved results, thereby reducing the influence of less relevant entities and relations on LLMs.

In addition to directly adjusting the textual information in prompts (i.e., hard prompts), some methods also apply soft prompts to enhance the output of LLMs. Soft prompting refers to a technique where continuous embeddings are prepended to the input sequence to guide LLMs' behavior. These embeddings are often learned or optimized specifically for a task, enabling the model to incorporate task-relevant information without modifying the parameters of LLMs. Soft prompts can be generated randomly, but common methods obtain soft prompts by encoding the graph in order to offer the structural information of the graph to LLMs. For example, He et al. [20] and Hu et al. [22] utilize GNNs to encode the retrieved subgraph as soft prompt and train them on a specific dataset. During the training process, the parameters of LLMs are frozen and only the parameters of the GNNs are updated.

IV. Evaluation

Evaluating knowledge graph RAG systems requires a robust framework to assess both retrieval and generation quality, as well as the effectiveness of incorporating knowledge graphs. This section outlines the benchmarks, metrics, experimental setups and methods utilized in the evaluation of current methods.

A. Benchmarks

Benchmarks are divided into downstream task datasets and special tests. The most common downstream task of the RAG system is question answering (QA). For open-domain question answering, where data can be obtained from open web pages and documents, there are datasets such as WikiQA [29], ODSQA [30] and IfQA [31], etc. And for knowledge base question answering (KBQA), where questions usually belong to a specific knowledge graph, and answers usually involve operations between entities, relations, or entity sets in the knowledge graph, some public knowledge graphs have been established such as Freebase [32]. There are also many KBQA datasets based on these knowledge graphs, such as WebQSP [33], HotpotQA [34] and FreebaseQA [35], etc. Benchmark tests are performance tests designed specifically for RAG systems. For example, He et al. [20] introduce a diverse benchmark targeted at real-world graph question answering, filling a crucial research gap. Jin et al. [19] construct a benchmark dataset called GRBENCH to support the development of methodology and facilitate the evaluation of the proposed models. It contains 1,740 questions that can be answered by 10 graphs from 5 domains.

B. *Metrics*

The evaluation metrics for the knowledge graph RAG systems can be divided into three parts: retrieval evaluation, generation evaluation, and graph evaluation. In terms of retrieval, metrics such as Recall@k, Mean Reciprocal Rank (MRR), and Normalized Discounted Cumulative Gain (NDCG) are standard for assessing how well relevant knowledge graph nodes or triples are retrieved based on a query. Some studies evaluate the performance of retrieval systems by analyzing the ratio between answer coverage and the size of the retrieved subgraph. For generation, standard metrics like BLEU, ROUGE, and METEOR are used to evaluate the fluency and relevance of generated responses. For open-domain tasks, F1 score and Exact Match (EM) are often utilized. In CSQA, Accuracy is the most commonly used evaluation metric. Specific to knowledge graph RAG systems, evaluating the model's effective use of graph structure is essential. Metrics like Graph Coverage (percentage of retrieved nodes that contribute relevant context) and Graph Relevance Score (weighting relevant entities higher) are increasingly utilized.

C. *Evaluation Mode*

The evaluation modes are mainly divided into manual evaluation and automatic evaluation. Manual evaluation usually refers to the subjective evaluation of some semantic indicators of LLM answers by humans, such as the completeness, comprehensiveness, and whether hallucinations occur in the answers. For example, Xu et al. [36] recruited graduate students to manually annotate the responses generated by LLMs identifying indicators that cannot be automatically derived, such as explain completeness, explain redundancy, perspective mistake, and process mistake.

In terms of automatic evaluation, in addition to the indicators in diverse benchmarks, many works utilize LLMs to evaluate the quality of RAG systems. For instance, GraphRAG [5] and LightRAG [7] utilize LLMs to evaluate the answers generated by RAG in terms of comprehensiveness, diversity, empowerment, and directness. Specifically, they group the answers generated by different RAGs into two groups, and then utilize the same model to determine which group of answers is better in these four metrics, and count the winning rates of different RAG systems.

V. Prospects

Although knowledge graphs have greatly improved the performance of RAG systems, there are still some challenges to be faced in this field. This section will discuss some of the current problems in this field and point out possible research directions in the future.

A. *Construction Flaws of KGs*

Undoubtedly, the performance of RAG is greatly affected by the quality of the knowledge graph. In recent studies, constructing knowledge graphs by prompting LLMs has become a common approach. However, this method can sometimes result in suboptimal graph structures, including ungrammatical entity names, multiple entities representing the same concept, and relationships with incorrect semantics. Therefore, optimizing the knowledge extraction process or exploring appropriate post-filtering and entity linking techniques is essential. An alternative solution involves utilizing document-relation graphs to bypass the knowledge extraction process. In this scenario, it is crucial to investigate methods for more effectively retrieving query-relevant entities from documents.

B. *Exploration of Heterogeneous and Dynamic Graphs*

Heterogeneous graphs, characterized by multiple types of nodes and edges, are common in domains like social networks, biomedical research, and recommendation systems. Extending RAG to handle such graphs could enable more nuanced reasoning by incorporating the semantics of diverse relationships and entities. For instance, incorporating node and edge type information directly into the retrieval and generation processes could improve the contextual relevance of outputs. Dynamic graphs, which evolve over time with changing nodes, edges, and attributes, introduce challenges such as maintaining consistency and capturing temporal patterns. RAG can be adapted to these scenarios by integrating temporal graph models or incremental graph retrievers, allowing it to handle evolving knowledge bases or streaming data effectively. This could be particularly impactful in real-time applications like event analysis, financial modeling, or adaptive question answering systems.

C. *Multimodal Integration*

Integrating multimodal data into RAG frameworks offers opportunities for richer and more comprehensive reasoning. By incorporating textual, visual, auditory, and other data modalities, RAG systems can address complex, cross-modal questions, such as interpreting diagrams or analyzing multimedia content. Challenges include designing retrieval mechanisms and generative models that effectively fuse heterogeneous data formats while maintaining semantic coherence. Future research could focus on cross-modal embeddings and multimodal graph structures to unify diverse data representations, enabling RAG to deliver contextually enriched and modality-aware responses, with applications in fields like medical diagnosis, interactive learning, and multimedia generation.

VI. Conclusion

This survey provides a comprehensive overview of integrating knowledge graphs into Retrieval-Augmented Generation systems, addressing their workflow and advantages in handling complex relationships, reducing noise, and enabling dynamic updates. Despite these benefits, challenges remain in constructing high-quality graphs, designing scalable retrieval mechanisms, and effectively utilizing graph structures during generation. Future research should focus on optimizing graph construction, exploring retrieval techniques for diverse graphs, and incorporating multimodal data, paving the way for more robust and versatile RAG systems.

References

[1] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin *et al.*, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *arXiv preprint arXiv:2311.05232*, 2023.

[2] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[3] J. Chen, H. Lin, X. Han, and L. Sun, "Benchmarking large language models in retrieval-augmented generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 17 754–17 762.

[4] Y. Kuratov, A. Bulatov, P. Anokhin, D. Sorokin, A. Sorokin, and M. Burtsev, "In search of needles in a 10m haystack: Recurrent memory finds what llms miss," *arXiv preprint arXiv:2402.10790*, 2024.

[5] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson, "From local to global: A graph rag approach to query-focused summarization," *arXiv preprint arXiv:2404.16130*, 2024.

[6] B. Peng, Y. Zhu, Y. Liu, X. Bo, H. Shi, C. Hong, Y. Zhang, and S. Tang, "Graph retrieval-augmented generation: A survey," *arXiv preprint arXiv:2408.08921*, 2024.

[7] Z. Guo, L. Xia, Y. Yu, T. Ao, and C. Huang, "Lightrag: Simple and fast retrieval-augmented generation," *arXiv preprint arXiv:2410.05779*, 2024.

[8] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[9] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[10] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[11] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang *et al.*, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024.

[12] Z. Nussbaum, J. X. Morris, B. Duderstadt, and A. Mulyar, "Nomic embed: Training a reproducible long context text embedder," *arXiv preprint arXiv:2402.01613*, 2024.

[13] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *COLING 1992 volume 2: The 14th international conference on computational linguistics*, 1992.

[14] S. Brin, "Extracting patterns and relations from the world wide web," in *International workshop on the world wide web and databases*. Springer, 1998, pp. 172–183.

[15] D. Sanmartin, "Kg-rag: Bridging the gap between knowledge and creativity," *arXiv preprint arXiv:2405.12035*, 2024.

[16] Y. Wang, N. Lipka, R. A. Rossi, A. Siu, R. Zhang, and T. Derr, "Knowledge graph prompting for multi-document question answering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 17, 2024, pp. 19 206–19 214.

[17] Z. Li, Q. Guo, J. Shao, L. Song, J. Bian, J. Zhang, and R. Wang, "Graph neural network enhanced retrieval for question answering of llms," *arXiv preprint arXiv:2406.06572*, 2024.

[18] S. Munikoti, A. Acharya, S. Wagle, and S. Horawalavithana, "Atlantic: Structure-aware retrieval-augmented language model for interdisciplinary science," *arXiv preprint arXiv:2311.12289*, 2023.

[19] B. Jin, C. Xie, J. Zhang, K. K. Roy, Y. Zhang, S. Wang, Y. Meng, and J. Han, "Graph chain-of-thought: Augmenting large language models by reasoning on graphs," *arXiv preprint arXiv:2404.07103*, 2024.

[20] X. He, Y. Tian, Y. Sun, N. V. Chawla, T. Laurent, Y. LeCun, X. Bresson, and B. Hooi, "G-retriever: Retrieval-augmented generation for textual graph understanding and question answering," *arXiv preprint arXiv:2402.07630*, 2024.

[21] S. Li, Y. Gao, H. Jiang, Q. Yin, Z. Li, X. Yan, C. Zhang, and B. Yin, "Graph reasoning for question answering with triplet retrieval," *arXiv preprint arXiv:2305.18742*, 2023.

[22] Y. Hu, Z. Lei, Z. Zhang, B. Pan, C. Ling, and L. Zhao, "Grag: Graph retrieval-augmented generation," *arXiv preprint arXiv:2405.16506*, 2024.

[23] C. Mavromatis and G. Karypis, "Gnn-rag: Graph neural retrieval for large language model reasoning," *arXiv preprint arXiv:2405.20139*, 2024.

[24] B. J. Gutiérrez, Y. Shu, Y. Gu, M. Yasunaga, and Y. Su, "Hipporag: Neurobiologically inspired long-term memory for large language models," *arXiv preprint arXiv:2405.14831*, 2024.

[25] H. Sun, T. Bedrax-Weiss, and W. W. Cohen, "Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text," *arXiv preprint arXiv:1904.09537*, 2019.

[26] P.-C. Lo and E.-P. Lim, "Contextual path retrieval: A contextual entity relation embedding-based approach," *ACM Transactions on Information Systems*, vol. 41, no. 1, pp. 1–38, 2023.

[27] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, H.-Y.

Shum, and J. Guo, "Think-on-graph: Deep and responsible reasoning of large language model with knowledge graph," *arXiv preprint arXiv:2307.07697*, 2023.

[28] J. Kim, Y. Kwon, Y. Jo, and E. Choi, "Kg-gpt: A general framework for reasoning on knowledge graphs using large language models," *arXiv preprint arXiv:2310.11220*, 2023.

[29] Y. Yang, W.-t. Yih, and C. Meek, "Wikiqa: A challenge dataset for open-domain question answering," in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 2013–2018.

[30] C.-H. Lee, S.-M. Wang, H.-C. Chang, and H.-Y. Lee, "Odsqa: Open-domain spoken question answering dataset," in *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2018, pp. 949–956.

[31] W. Yu, M. Jiang, P. Clark, and A. Sabharwal, "Ifqa: A dataset for open-domain question answering under counterfactual presuppositions," *arXiv preprint arXiv:2305.14010*, 2023.

[32] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1247–1250.

[33] W.-t. Yih, M. Richardson, C. Meek, M.-W. Chang, and J. Suh, "The value of semantic parse labeling for knowledge base question answering," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2016, pp. 201–206.

[34] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning, "Hotpotqa: A dataset for diverse, explainable multi-hop question answering," *arXiv preprint arXiv:1809.09600*, 2018.

[35] K. Jiang, D. Wu, and H. Jiang, "Freebaseqa: A new factoid qa data set matching trivia-style question-answer pairs with freebase," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 318–323.

[36] F. Xu, Q. Lin, J. Han, T. Zhao, J. Liu, and E. Cambria, "Are large language models really good logical reasoners? a comprehensive evaluation from deductive, inductive and abductive views," *arXiv preprint arXiv:2306.09841*, 2023.