

# Efficient Knowledge Graph Construction and Retrieval from Unstructured Text for Large-Scale RAG Systems

CONGMIN MIN, SAP, USA

RHEA MATHEW, SAP, USA

JOYCE PAN\*, SAP, USA

SAHIL BANSAL\*, SAP, USA

ABBAS KESHAVARZI\*, SAP, USA

AMAR VISWANATHAN KANNAN, SAP, USA

We propose a scalable and cost-efficient framework for deploying Graph-based Retrieval-Augmented Generation (GraphRAG) in enterprise environments. While GraphRAG has shown promise for multi-hop reasoning and structured retrieval, its adoption has been limited by the high computational cost of constructing knowledge graphs using large language models (LLMs) and the latency of graph-based retrieval. To address these challenges, we introduce two core innovations: (1) a dependency-based knowledge graph construction pipeline that leverages industrial-grade NLP libraries to extract entities and relations from unstructured text—completely eliminating reliance on LLMs; and (2) a lightweight graph retrieval strategy that combines hybrid query node identification with efficient one-hop traversal for high-recall, low-latency subgraph extraction. We evaluate our framework on two SAP datasets focused on legacy code migration and demonstrate strong empirical performance. Our system achieves up to 15% and 4.35% improvements over traditional RAG baselines based on LLM-as-Judge and RAGAS metrics, respectively. Moreover, our dependency-based construction approach attains 94% of the performance of LLM-generated knowledge graphs (61.87% vs. 65.83%) while significantly reducing cost and improving scalability. These results validate the feasibility of deploying GraphRAG systems in real-world, large-scale enterprise applications without incurring prohibitive resource requirements paving the way for practical, explainable, and domain-adaptable retrieval-augmented reasoning.

**CCS Concepts:** • Information systems → Question answering; Document filtering; Top-k retrieval in databases.

**Additional Key Words and Phrases:** Knowledge Graph, Retrieval Augmented Generation (RAG), Dependency Parsing, Hybrid Search, Scalable GraphRAG, GraphRAG, Code Migration

## 1 Introduction

Retrieval-Augmented Generation (RAG) has emerged as a practical framework for enhancing large language models (LLMs) by grounding their outputs in external knowledge sources. In a standard RAG pipeline, a user query triggers the retrieval of semantically relevant passages from a document corpus using dense-vector techniques. These retrieved passages are then fed to the LLM as contextual input, anchoring its responses in factual content. This architecture helps reduce hallucinations and enables the model to stay current with evolving information—without the need for expensive model retraining [27]. In enterprise settings, RAG allows organizations to integrate proprietary data so that generated responses align with the latest domain-specific knowledge [16].

---

\*All authors contributed equally to this research.

---

Authors' Contact Information: Congmin Min, SAP, Palo Alto, California, USA, congmin.min@sap.com; Rhea Mathew, SAP, Palo Alto, California, USA, rheamathew@sap.com; Joyce Pan, SAP, Palo Alto, California, USA, joyce.pan01@sap.com; Sahil Bansal, SAP, Palo Alto, California, USA, sahil.bansal01@sap.com; Abbas Keshavarzi, SAP, Palo Alto, California, USA, abbas.keshavarzi@sap.com; Amar Viswanathan Kannan, SAP, Palo Alto, California, USA, amar.viswanathan.kannan@sap.com.

While RAG performs well for straightforward fact-based queries, it often fails to deliver coherent results for more complex tasks that require reasoning across multiple documents [37]. For example, questions involving policy dependencies, multi-system workflows, or legacy code migration frequently span several pieces of content and require connecting implicit relationships. In such cases, standard RAG pipelines tend to return isolated snippets without an understanding of how those pieces relate. This limits the model’s ability to synthesize a logically coherent answer, often resulting in brittle or incomplete outputs—particularly in high-stakes enterprise environments.

Modern ERP systems, such as those used for finance, procurement, HR, and manufacturing, generate vast volumes of structured and unstructured data across interconnected modules. Enterprise queries often involve reasoning over configuration rules, transactional dependencies, change logs, and migration guides that are distributed across documents and systems. For example, assessing the impact of a custom code migration in SAP’s S/4HANA may require linking legacy ABAP functions with deprecation reports, compatibility matrices, and policy guidelines. Traditional RAG systems are ill-suited for this kind of multi-hop, relational reasoning. Graph-based retrieval provides a natural fit for these scenarios, as it captures structured dependencies and enables traversal-based querying across linked entities making GraphRAG a promising solution for ERP-related applications.

Graph-based RAG (GraphRAG) addresses the limitations of traditional RAG by constructing a structured knowledge graph from the source corpus to enable semantically aware retrieval and multi-hop reasoning. During indexing, documents are processed to extract entities and their relations, which are then stored as nodes and edges in a knowledge graph. At query time, the system retrieves both individual passages and relevant subgraphs that encode logical connections across documents. This structure enables traversal over semantically meaningful paths, allowing the system to assemble chains of evidence and deliver coherent, multi-step responses.

However, deploying GraphRAG in enterprise settings introduces three core challenges:

- **Computational cost of graph construction.** Building a knowledge graph at enterprise scale requires large-scale entity and relation extraction. When this process relies on LLMs or heavyweight NLP pipelines, it incurs significant GPU or CPU costs, leading to high latency and limited refresh frequency for dynamic content.
- **Scalability limitations.** As document collections grow, maintaining and updating the knowledge graph becomes increasingly difficult. Many existing approaches do not scale beyond hundreds of thousands of nodes and lack efficient mechanisms for incremental updates or distributed storage.
- **Retrieval inefficiency.** Querying large graphs for relevant subgraphs often introduces latency that hampers interactive use cases. Even optimized graph databases can struggle with real-time performance when executing multi-hop traversals or subgraph ranking.

### 1.1 Contributions

In this paper, we propose a GraphRAG framework tailored for enterprise-scale use cases. Our contributions are as follows:

- We present a dependency-based knowledge graph construction pipeline using industrial-grade NLP libraries, eliminating reliance on LLMs and reducing the cost barrier for scalable deployment.
- We introduce a lightweight graph retrieval strategy that combines hybrid query node identification with efficient one-hop traversal to retrieve high-recall, semantically relevant subgraphs.

- We are the first, to the best of our knowledge, to apply GraphRAG to a real-world legacy code migration task, demonstrating significant improvements over dense-only retrieval in both qualitative and quantitative evaluations.

These contributions demonstrate how GraphRAG can deliver explainable, accurate, and scalable retrieval-augmented reasoning in complex enterprise environments. We contextualize our work in relation to prior efforts in retrieval, graph-based reasoning, and large-scale knowledge integration in the sections that follow.

## 2 Related Work

Retrieval-Augmented Generation (RAG) was first introduced by [27], combining a dense-vector retriever over Wikipedia with a Sequence-to-Sequence (seq2seq) language model. This integration improved question answering and generation quality over purely parametric models. However, subsequent analyses [4, 5] identified systemic limitations across multiple domains, including enterprise-specific concerns such as security, interpretability, and scalability. These works also proposed frameworks to assess enterprise-readiness of RAG pipelines.

To address these gaps, the GraphRAG paradigm was introduced, embedding a structured knowledge graph between the retrieval and generation stages [20]. Microsoft’s GraphRAG demonstrated that constructing entity-relation graphs from retrieved passages and summarizing them into semantic communities improved QA performance. Building on this foundation, recent innovations—such as LightRAG, FastGraphRAG, and MiniRAG—have focused on designing lightweight, efficient graph representations to accelerate retrieval [1, 14, 18]. Others, like HyperTree Planning [17], use hierarchical graph-guided reasoning for multi-step inference. A recent survey [35] categorizes GraphRAG strategies by construction and integration type, while application-focused frameworks like “From Local to Global” tailor graph context aggregation for summarization [12].

The scalability of GraphRAG depends heavily on the underlying graph infrastructure. Systems like GraphScope [15] and GraphScope Flex [21] support distributed graph analytics and GNN training, achieving near-linear speedups on trillion-edge workloads via modular execution. AliGraph [44] further advances large-scale graph training with optimized sampling and storage, reporting 12× speedup over baseline systems.

Despite these advances, scalable and real-time subgraph retrieval remains a key challenge. For instance, GRAG [23] uses divide-and-conquer strategies to segment large graphs, but this can introduce latency as the number of subgraphs grows. The RGL library [28] addresses parts of this challenge by integrating graph indexing, dynamic node retrieval, and subgraph construction—demonstrating substantial speed gains. However, its limited support for diverse graph database backends still constrains its generalizability in enterprise environments. Similarly, efficient syntactic methods like dependency parsing offer scalable alternatives for lightweight graph construction [6, 31, 33], which we leverage in our pipeline.

While prior work has explored source prioritization [25] and retrieval evaluation at scale [34], these approaches are largely confined to traditional RAG architectures. In contrast, our GraphRAG framework introduces a semantically grounded, structured retrieval layer built on top of domain-agnostic knowledge graphs. This design supports accurate, explainable, and scalable response generation for complex enterprise queries. Importantly, it complements large-scale graph systems through our lightweight NLP-driven construction and retrieval stack—validating its practical deployability in industrial contexts.

## 2.1 Graph Representations

Graph Neural Networks (GNNs) have been applied to encode graph structure and generate node embeddings for retrieval [7], but their inference speed is a bottleneck in large-scale systems. The computational cost of message passing across millions of nodes and edges hinders their real-time applicability in low-latency enterprise settings—unless heavily pruned or cached. To improve scalability, alternative algorithms have emerged. Personalized PageRank (PPR) offers a lightweight, proximity-based node ranking mechanism [42], effective in small-world graphs where relevant information lies within a few hops. However, executing PPR on-the-fly for every query is still computationally expensive in high-throughput environments, and pre-computing vectors becomes infeasible as graphs evolve [11, 29, 45]. We currently implement a basic PPR module and are developing an optimized version to better support real-time workloads. Another promising direction involves community detection to restrict retrieval to semantically dense subgraphs. Microsoft’s GraphRAG [12] employs this technique, leveraging modularity-based algorithms like Leiden [38] to pre-select relevant graph regions. This allows retrieval to be focused on a small number of communities, improving both latency and answer quality. Taken together, these retrieval strategies—GNNs, PPR, and community-based selection—each contribute partial solutions to the retrieval bottleneck in GraphRAG. However, integrating them into a unified, scalable, enterprise-ready pipeline remains an open challenge. Our work builds toward this vision by prioritizing efficiency, adaptability, and explainability in every layer of the GraphRAG architecture.

## 3 Methodology

Our system is built on top of the following two steps: (i) An interchangeable knowledge graph framework that supports both LLM based KG generation and a lightweight dependency parser based KG construction , and (ii) a cascaded retrieval system that combines one-hop graph traversal with dense vector-based node re-ranking.

The knowledge graph is constructed using one of two interchangeable pipelines. (i) a high-quality but computationally expensive LLM-based extractor, or (ii) a lightweight, cost-effective dependency-parser-based builder. The resulting graph is stored in a dedicated knowledge graph store for downstream retrieval.

To motivate our graph construction strategy, we draw upon dependency grammar [10], which posits that a sentence’s syntactic structure can be represented as a graph of binary *head-dependent* relations. For example, in the sentence *The developer refactored the Z-report for S/4HANA*, *refactored* is the head verb, while *developer*, *Z-report*, and *S/4HANA* are its dependents. Similarly, in *The custom function module was flagged as incompatible*, *flagged* serves as the head, linking *function module* and *incompatible*. When extracted from SAP Custom Code Migration (CCM) logs and documentation, such sentence structures can be connected into a local knowledge graph reflecting relationships such as (i) developer refactored Z-report, (ii) function module flagged incompatible, and (iii) Z-report adapted for S/4HANA .

At query time, we apply a two-stage retrieval strategy. First, we conduct a high-recall one-hop graph traversal to identify candidate nodes. Next, we apply a dense vector-based re-ranking step using OpenAI embeddings and cosine similarity to refine the result set. The selected subgraph, along with relevant source text chunks and extracted query entities, is then passed to an LLM summarizer to generate the final, focused response. Our retrieval approach aligns with the classical cascaded architecture in information retrieval (IR), where an initial recall-oriented stage (e.g., BM25 or dense vector search) is followed by a precision-oriented neural re-ranker [2, 30, 32]. Drawing from small-world connectivity theory [41], our one-hop traversal effectively retrieves semantically related nodes while keeping the candidate set size tractable—crucial for scaling to large enterprise graphs.

In the following sections, we present our full end-to-end pipeline: beginning with knowledge graph construction, continuing through targeted retrieval, and concluding with query-focused summarization.

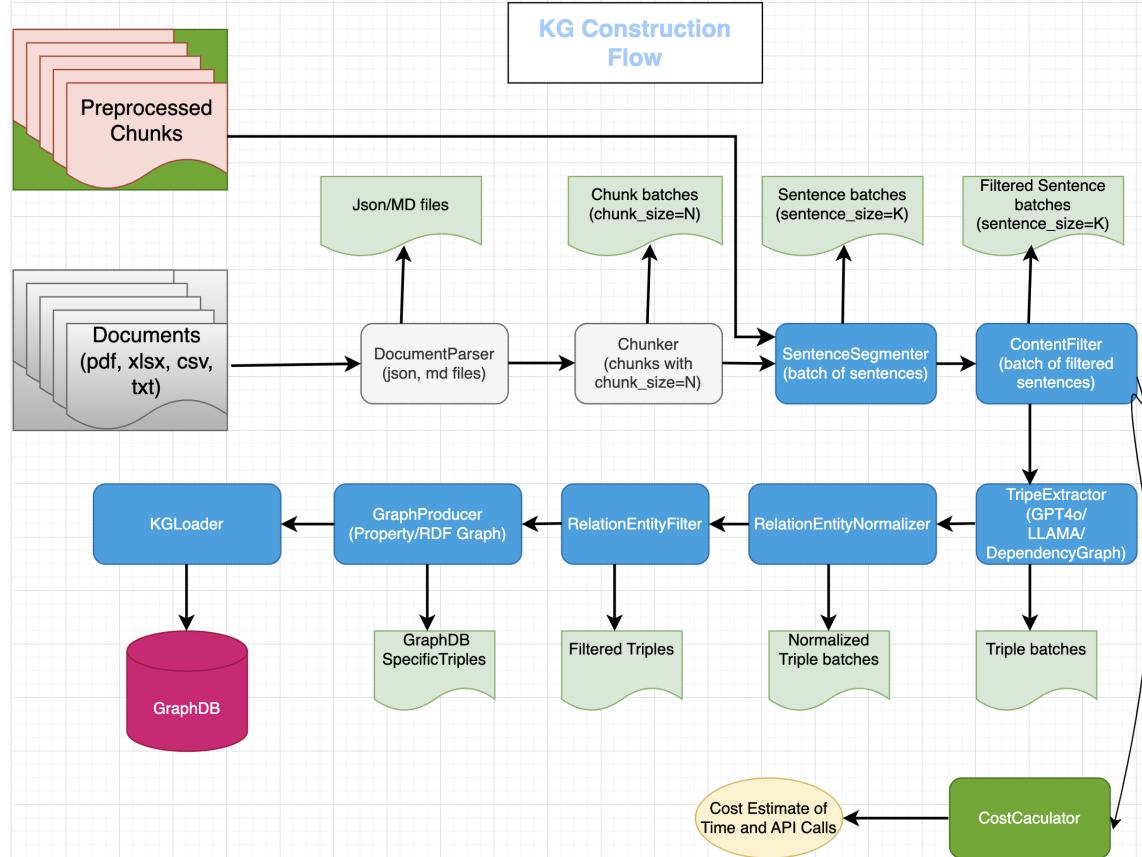


Fig. 1. Multi-model KG Construction Pipeline

### 3.1 Knowledge Graph Construction

Figure 1 illustrates our multi-model knowledge graph construction pipeline, which supports both an LLM-based extraction path and a lightweight dependency-parser-based alternative. Input documents pass through a series of preprocessing and filtering stages before triples are extracted, normalized, and materialized in the target graph store.

The key components of the construction pipeline are described below:

**DocumentParser:** Input documents arrive in a variety of formats: PDF, HTML, XLSX, CSV, etc. We utilize the open-source Docling library<sup>1</sup> to convert files in any format to a unified intermediate representation. This representation retains layout, tables, and structural metadata, facilitating downstream GraphRAG processing. The final parsed output is provided in either JSON or Markdown (MD) format.

<sup>1</sup><https://github.com/docling-project/docling>

**HybridChunker:** Building on the insights of Hearst et al. [22], we adopt a hierarchical chunking strategy that prioritizes discourse-level boundaries. Our two-stage approach first splits documents at Markdown headers to preserve semantic cohesion, and then applies character-level splitting when sections exceed predefined size limits. The chunking configuration uses a maximum size of 2048 characters with 200-character overlap. Our implementation is based on LangChain’s RecursiveCharacterTextSplitter<sup>2</sup> and includes engineering enhancements such as whitespace-aware sizing to better support enterprise-scale workflows.

**SentenceSegmenter:** Each text chunk is segmented into individual sentences using language-specific delimiters. This serves two primary purposes (i) LLMs often degrade in performance with longer input lengths [26, 36], and (ii) sentence-level units are more amenable to syntactic parsing, allowing us to filter content based on linguistic structure. We employ the SpaCy<sup>3</sup> library for parsing and use part-of-speech tags to filter out sentences lacking verbs. This reduces the amount of LLM calls during downstream entity/relation extraction. For efficiency, we batch sentences for concurrent API calls using a sliding window configuration having a sentence size of 3 and sentence overlap of 1.

**ContentFilter:** From the outputs of the sentence segmenter, we filter content that does not have verbs in its syntactic structure. This allows for efficient pruning of content. Such kind of strategies are prevalent in the community and has shown to have great success [40].

**TripleExtractor:** In the triple extractor we allow the option of utilizing either commercial LLMS (GPT 4o and Sonnet) or a dependency parser based approach. In this paper we utilize the dependency parser approach. But the system has an option of switching based on the costs calculation. Depending on the dataset size, we will choose between GPT-4o and dependency graph models for different use cases. Specifically, we assessed the same dataset using both GPT-4o and our dependency-graph model.

*3.1.1 Dependency Parsing.* This dependency parsing approach combines traditional syntactic parsing with specialized heuristics for technical text, creating a robust system for extracting structured knowledge from unstructured text. One particularly interesting property of this approach is that it is domain agnostic, meaning it can be applied across a wide range of domains without requiring domain-specific training or customization, making it highly adaptable for diverse text. We leverage SpaCy’s dependency parser to extract entities and relations for two considerations (i) SpaCy is built for industrial use and offers high-speed performance. (ii) It includes a state-of-the-art dependency parser well-suited for open-ended information extraction. We also evaluated other lightweight models like the specialized in information extraction; however, they performed poorly on open-ended information extraction tasks. For “*SAP launched Joule for Consultants*”, SpaCy generates the parse tree shown in Figure 2. It’s important to note that the dependency parser itself does not produce knowledge triples required for building a knowledge graph. Instead, it creates syntactic trees, and it is our responsibility to implement extraction logic to convert the trees to structured knowledge triples. Our DependencyExtractor does exactly that and identifies knowledge triples *{SAP, launch, Joule\_for\_Constants}* from the given parse tree. It performs sophisticated dependency parsing logic to extract subject-relation-object triples from text. Below is a high-level overview of the key steps involved:

- Noun Phrase Extraction and Cleaning
- Verb Processing and Relation Extraction
- Subject and Object Identification
- Special Pattern Recognition

---

<sup>2</sup>[https://api.python.langchain.com/en/latest/character/langchain\\_text\\_splitters.character.RecursiveCharacterTextSplitter.html](https://api.python.langchain.com/en/latest/character/langchain_text_splitters.character.RecursiveCharacterTextSplitter.html)

<sup>3</sup><https://spacy.io/>

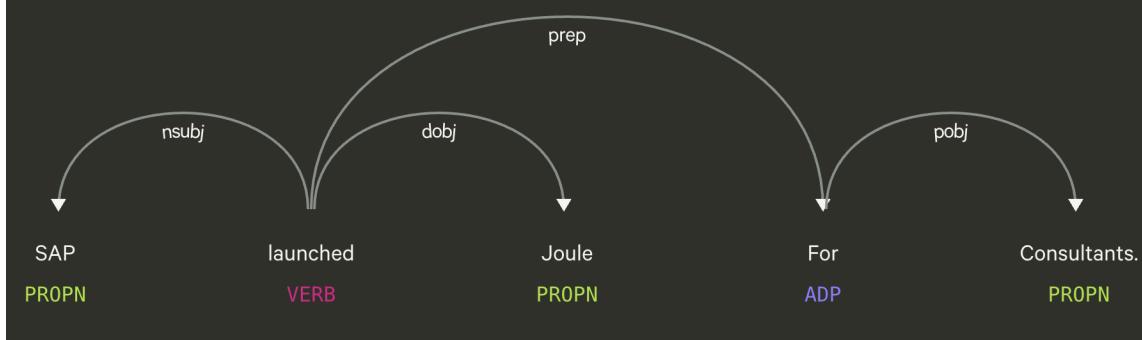


Fig. 2. SpaCy Generated Parse Tree

- Triple Formation and Post-Processing
- Advanced Contextual Analysis

**CostCalculator:** We developed a cost calculator to efficiently estimate the cost of API calls involved while building the Knowledge Graph. This gets updated based on the pricing per token of the commercial LLMs. This ensures full understanding of the true cost in building a Knowledge Graph.

**EntityRelationNormalizer:** Natural language text is noisy, and technical documents often contain special characters or symbols that are incompatible with graph storage systems. For instance, some graph databases (DB) do not allow colons (':') in entity labels, and RDF triples cannot include unescaped null characters. The EntityRelationNormalizer performs two key functions (i) it enables deduplication by normalizing variations of the same entity and relation to be merged into one and (ii) it standardize entity and relation names to ensure compatibility with the graph database.

**RelationEntityFilter:** Our knowledge graph pipeline accommodates both schema-less and schema-based knowledge graph construction. Unlike open-ended information extraction, constructing a schema-guided knowledge graph benefits from a pre-defined schema that outlines a specific set of entities and relationships, as defined by domain experts or stakeholders with deep expertise. We use the RelationEntityFilter to post-process the entities and relations extracted by our model, ensuring they conform to this established schema.

**GraphProducer:** GraphProducer module accepts generic triples generated by the model and transforms them into a graph format compatible with the target graph database. Currently, we convert knowledge triples into property graph format. Future work includes extending support to RDF triple conversion.

**KGLoader:** KGLoader accepts input in a specified graph data format and loads it into the designated graph database. We have implemented different loaders for different destinations for graph visualization, analysis and production.

### 3.2 Efficient Graph Retrieval Process

Figure 3 above illustrates the major components in our indexing and retrieval pipeline. During indexing, the knowledge graph is stored in both Vector DB and Graph DB. For our experiments, we use the open-source library Milvus [39] for storing embeddings and the high-performance iGraph [9] to store the graph in memory. Milvus stores nodes, chunks and relation embeddings for fast similarity lookup at query time, and iGraph stores nodes and edges for fast traversal. The following sections describe the major components for the retrieval processes.

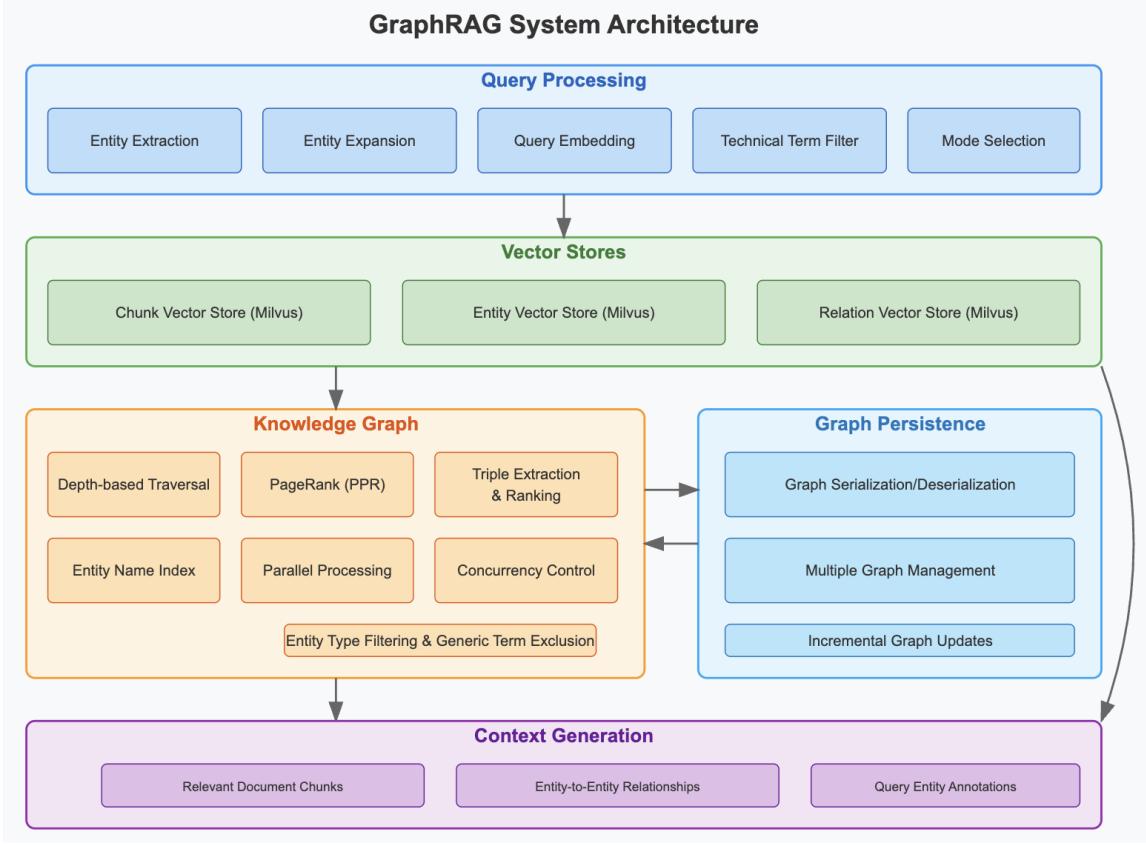


Fig. 3. GraphRAG Indexing and Retrieval Architecture

**3.2.1 Query Entity Identification.** In contrast to other GraphRAG methods [3, 19, 24] that rely on LLMs for entity identification, we employ an optimized variant of SpaCy’s noun phrase extractor to efficiently pinpoint key concepts within the query. Additionally, we conduct a similarity search between the full query and node embeddings to retrieve the top- $k$ , where  $k = 5$  relevant nodes from the graph. The entities obtained from both approaches are then merged and used as seed nodes for relation extraction.

**3.2.2 Graph Query Execution.** Starting from seed query nodes, we use case insensitive exact match to query the graph for relevant relations. Once a node is matched with a query node, it performs 1-hop traversal of all neighbors and filtered by a neighbor controlling parameter  $random\_k\_relations$ . For small to medium size graph,  $random\_k\_relations = 100$  is good, for larger ones, we set it to 200 akin to Yasunaga et al [43].

**3.2.3 Relevance Ranking and Context Selection.** Once the candidate relations are obtained through case insensitive exact match and graph traversal, they are split into two groups: entity-to-entity relations and entity-to-chunk relations. Both chunk and relation embeddings are retrieved from the Milvus vector DB, which are then used to compute cosine similarity with the query. Chunks and relations are then sorted by similarity scores, and top- $k$  chunks and top- $k * 2$

relations are returned. In selecting top- $k$  chunks, we also provide an option of enabling hybrid search which combines semantic search and graph search together through Reciprocal Rank Fusion [8] to produce the best results.

**3.2.4 Context Integration with LLM.** Once the top- $k$  chunks and top- $k$  relations are produced, we send them along with query entities as the context for LLM to consume and generate answers. Context is a dictionary with three keys:  $\text{Context} = \{\text{"chunks"} : \text{chunk\_list}, \text{"relations"} : \text{relation\_list}, \text{"entity"} : \text{entity\_list}\}$ , which provides a much richer context than standard RAG alone.

## 4 Experiments

### 4.1 Datasets

We adapt CCM as a domain-specific use case for our model. The datasets used in this study comprise three main components: (i) a grounded resource corpus, (ii) the CCM Chat test set, and (iii) the CCM Code Proposal test set.

The CCM resource corpus consists of 550 PDF documents, including SAP Cookbooks and Notes related to code migration. These documents are pre-processed into approximately 2000 text chunks, each with a length of 3000 characters and an overlap of 500 characters. This processed corpus serves as the foundation for both knowledge graph construction and dense vector representation in our experiments.

The two test datasets are designed to evaluate different aspects of the system. CCM Chat includes 150 question-answer pairs focused on code migration topics, such as error analysis and implementation differences before and after migration. CCM Code Proposal comprises 200 legacy code examples, each containing the original (pre-migration) code alongside the corrected (post-migration) version.

Using the CCM resource corpus, our method yields a knowledge graph consisting of 39155 nodes, 47613 entity-to-entity relations, 63681 entity-to-chunk relation, resulting in an average node degree of 1.52 and a highest degree of 236.

**4.1.1 Evaluation Methodology.** : For *CCM Chat Evaluation*. we employ two complementary evaluation techniques:

(i) **Coverage Measurement** and (ii) **RAGAS Scores**.

*Coverage Measurement* uses an LLM-based classifier to compare the generated response against a reference (ground truth) answer. The LLM is prompted to assign a discrete semantic coverage score: (i) **0** if the answer fails to cover any part of the ground truth, (ii) **0.5** if it partially captures key information, and (iii) **1** if it fully aligns with the reference answer. An overall performance metric is computed as a weighted average:

$$\text{Weighted Average Score} = (0.5 \times \%_{0.5} + 1.0 \times \%_{1.0}) \times 100\%$$

where  $\%_{0.5}$  and  $\%_{1.0}$  represent the proportions of responses assigned scores of 0.5 and 1.0, respectively.

*RAGAS Evaluation* [13] assesses both the retrieval quality and generative accuracy using three metrics: (i) **Context Precision**, which measures the proportion of retrieved chunks that are relevant to the question; (ii) **Faithfulness**, which quantifies how much of the generated answer is grounded in the retrieved content; and (iii) **Answer Relevancy**, which evaluates how directly the generated response addresses the original query. This is computed by generating follow-up questions based on the response and comparing their cosine similarity to the original query—a higher similarity indicates stronger relevance.

Table 1. RAGAS based Evaluation on CCM Chat

Method	Context Precision	Faithfulness	Answer Relevancy	Avg.
Dense Vector (ada-002)	54.35%	77.18%	82.92%	71.48%
GraphRAG (GPT-4o)	63.82%	74.24%	89.43%	75.83%
GraphRAG (Dependency)	61.07%	72.76%	90.97%	74.93%

Table 2. LLM-as-a-Judge based Evaluation on CCM Chat

Method	No Cov. (0)	Partial Cov. (0.5)	Full Cov. (1)	Weighted Avg.
Dense Vector (ada-002)	40.29%	15.85%	42.88%	50.80%
GraphRAG (GPT-4o)	27.34%	13.67%	58.99%	65.83%
GraphRAG (Dependency)	27.34%	21.58%	51.08%	61.87%

For *CCM Code Proposal Evaluation*, we adopt an *LLM-as-a-Judge* framework to compare generated migration code with the ground truth. Each evaluation instance includes legacy input code, the reference migrated version, and two generated outputs—one produced using dense vector retrieval and the other via graph-based retrieval.

Evaluation proceeds in two stages: (i) **Pairwise Comparison**, where the LLM selects the more accurate candidate based on similarity to the ground truth; and (ii) **Dimensional Scoring**, where each generated output is rated across five criteria:

- **Syntax Correctness** — validity of the code’s syntax.
- **Logical Correctness** — alignment of program logic with the ground truth.
- **S/4HANA Compatibility** — adherence to S/4HANA APIs, libraries, and coding standards.
- **Optimization and Efficiency** — performance quality, including runtime and structural improvements.
- **Readability** — clarity of formatting, naming conventions, and maintainability.

## 4.2 Results and Analysis

**4.2.1 System Performance Result.** On **CCM Chat**, the evaluation results for *CCM Chat* are presented in Table 1 and Table 2. Both variants of GraphRAG, one using GPT-4o for triplet extraction and the other using dependency graph as triplet creation model, show at least 12% improvement in context precision score compared to dense vector retrieval. In terms of coverage measurement (abbreviated as No Cov., Partial Cov., and Full Cov. in Table 2), the *No Coverage* rate is reduced by 32% for both variants, while the *Full Coverage* rate increases by at least 19%.

Notably, the dependency graph-based GraphRAG model retains 94% of the GPT-4o variant’s performance in context precision. It achieves comparable results in *No Coverage* and reaches 86.6% of the GPT-4o variant’s performance in *Full Coverage* highlighting its strong performance with a lighter knowledge graph construction pipeline.

**CCM Code Proposal Results.** On *CCM Code Proposal* dataset, both GraphRAG variants outperform dense retrieval in terms of winning rate and average score, where average score measures an average across all five evaluation criteria. The GraphRAG created by dependency parsing achieves performance on par with GPT-4o variant, indicating that dependency graph-based GraphRAG is a strong alternative to LLM-based triplet extraction in retrieval tasks.

Table 3. LLM-as-a-Judge based Evaluation on CCM Code Proposal (GPT-4o-based)

Method	Winning Rate	Avg. Score (1–5)
Dense Vector (ada-002)	23%	3.48
GraphRAG (GPT-4o)	77%	4.04

Table 4. LLM-as-a-Judge based Evaluation on CCM Code Proposal (Dependency Graph-based)

Method	Winning Rate	Avg. Score (1–5)
Dense Vector (ada-002)	21.5%	3.43
GraphRAG (Dependency)	78.5%	4.03

Table 5. Cost Evaluation on GPT-4o-based KG Construction

Processing Step	Workload	No Parallel	2 Workers in Parallel
GPT-4o API Calls	$800,000 \times 7 \text{ s} = 5,600,000 \text{ s}$	~ 64.8 days	~ 32.4 days
Result Parse and Insert	$800,000 \times 0.1 \text{ s} = 80,000 \text{ s}$	~ 22.2 hours	~ 11.1 hours
Graph Post-processing	One time ~ 5 minutes	~ 5 minutes	~ 5 minutes
<b>Total</b>		~ 65.7 days	~ 33 days

*Qualitative Insight.* Our method is effective in retrieving content tied to key entities. For example, in *CCM chat*, given question "How do I handle custom code that references VBBS (summarized requirements) after the S/4HANA conversion?", dense retriever fails to extract content that specifically address VBBS table. In contrast, GraphRAG selects content explicitly mentioning "VBBS" and includes relevance sentences: "... If the VBBS is used in customer code, ... The solution is to create a view on VBBE ...". This illustrates our system's strength in extracting semantically focused context.

In *CCM Code Proposal*, from our error analysis we observe that the migrated code generated by dense vector retrieval exhibits more hallucinations and faulty function definitions compared to that generated using GraphRAG-based retrieval. For example, a fake function call "call function 'sd\_vbuk\_read\_from\_doc\_multi'" is generated by the dense vector.

## 5 Conclusion and Future Work

In this work, we present a scalable method for constructing enterprise-grade GraphRAG systems from unstructured text. To address key scalability challenges in real-world enterprise environments, our approach centers on two core components: (i) knowledge graph construction using efficient dependency parsing, and (ii) lightweight, hybrid subgraph retrieval to ensure low-latency query-time performance. We validate our framework on two SAP use cases, CCM Chat and CCM Code Proposal, and observe consistent performance improvements over a baseline RAG system. Notably, knowledge graphs generated using a robust, open-source dependency parser achieved performance comparable to GPT-4o, as measured by both LLM-as-a-Judge and RAGAS evaluation metrics.

Our approach offers a promising path for scaling GraphRAG systems by eliminating one of the most significant bottlenecks: dependence on large language models for knowledge graph construction. Nevertheless, two limitations warrant future investigation. First, while dependency parsing provides a lightweight and scalable method for extracting knowledge triples, it may miss context-dependent or implicit relations not directly expressed in surface syntax. Second, although our method demonstrates strong performance in SAP-specific domains, its generalizability to other settings

remains an open question. Future work includes evaluating the approach on broader public benchmarks such as HotpotQA to assess its applicability beyond enterprise use cases.

## 6 GenAI Usage Disclosure

We employed ChatGPT to assist in rephrasing certain sections of the paper for improved clarity. All core content, including research design, data analysis, and result interpretation, was conducted without the aid of generative AI tools.

## References

- [1] Amar Abane, Anis Bekri, and Abdella Battou. 2024. FastRAG: Retrieval Augmented Generation for Semi-structured Data. arXiv:2411.13773 [cs.NI] <https://arxiv.org/abs/2411.13773>
- [2] Omar Adjali, Olivier Ferret, Sahar Ghannay, and Hervé Le Borgne. 2024. Multi-Level Information Retrieval Augmented Generation for Knowledge-based Visual Question Answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 16499–16513.
- [3] Circlemind AI. 2025. fast-graphrag: Fast and Modular Graph-based RAG Framework. <https://github.com/circlemind-ai/fast-graphrag>. Accessed: 2025-05-22.
- [4] Scott Barnett, Stefanus Kurniawan, Srikanth Thudumu, Zach Brannelly, and Mohamed Abdelrazek. 2024. Seven failure points when engineering a retrieval augmented generation system. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*. 194–199.
- [5] Tilmann Bruckhaus. 2024. Rag does not work for enterprises. *arXiv preprint arXiv:2406.04369* (2024).
- [6] Razvan Bunescu and Raymond Mooney. 2005. A shortest path dependency kernel for relation extraction. In *Proceedings of human language technology conference and conference on empirical methods in natural language processing*. 724–731.
- [7] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*. ACM. doi:10.1145/3292500.3330925
- [8] Gordon V Cormack, Charles LA Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. 758–759.
- [9] Gábor Csárdi and Tamás Nepusz. 2006. The igraph software package for complex network research. *InterJournal, Complex Systems* 1695 (2006). <https://igraph.org>
- [10] Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal Dependencies. *Computational Linguistics* 47, 2 (June 2021), 255–308. doi:10.1162/coli\_a\_00402
- [11] Yuxin Dong, Shuo Wang, Hongye Zheng, Jiajing Chen, Zhenhong Zhang, and Chihang Wang. 2024. Advanced RAG Models with Graph Structures: Optimizing Complex Knowledge Reasoning and Text Generation. In *2024 5th International Symposium on Computer Engineering and Intelligent Communications (ISCEIC)*. IEEE, 626–630.
- [12] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).
- [13] Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. 2025. Ragas: Automated Evaluation of Retrieval Augmented Generation. arXiv:2309.15217 [cs.CL] <https://arxiv.org/abs/2309.15217>
- [14] Tianyu Fan, Jingyuan Wang, Xubin Ren, and Chao Huang. 2025. MiniRAG: Towards Extremely Simple Retrieval-Augmented Generation. arXiv:2501.06713 [cs.AI] <https://arxiv.org/abs/2501.06713>
- [15] Wenfei Fan, Tao He, Longbin Lai, Xue Li, Yong Li, Zhao Li, Zhengping Qian, Chao Tian, Lei Wang, Jingbo Xu, et al. 2021. GraphScope: a unified engine for big graph processing. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2879–2892.
- [16] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* 2 (2023), 1.
- [17] Runquan Gui, Zhihai Wang, Jie Wang, Chi Ma, Huiling Zhen, Mingxuan Yuan, Jianye Hao, Defu Lian, Enhong Chen, and Feng Wu. 2025. HyperTree Planning: Enhancing LLM Reasoning via Hierarchical Thinking. *arXiv preprint arXiv:2505.02322* (2025).
- [18] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation. (2024).
- [19] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2025. LightRAG: Simple and Fast Retrieval-Augmented Generation. arXiv:2410.05779 [cs.IR] <https://arxiv.org/abs/2410.05779>
- [20] Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, et al. 2024. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309* (2024).
- [21] Tao He, Shuxian Hu, Longbin Lai, Dongze Li, Neng Li, Xue Li, Lexiao Liu, Xiaojian Luo, Bingqing Lyu, Ke Meng, et al. 2024. Graphscope flex: Lego-like graph computing stack. In *Companion of the 2024 International Conference on Management of Data*. 386–399.

- [22] Marti A. Hearst. 1997. Text Tiling: Segmenting Text into Multi-paragraph Subtopic Passages. *Computational Linguistics* 23, 1 (1997), 33–64. <https://aclanthology.org/J97-1003/>
- [23] Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. 2024. Grag: Graph retrieval-augmented generation. *arXiv preprint arXiv:2405.16506* (2024).
- [24] Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. *Advances in Neural Information Processing Systems* 37 (2024), 59532–59569.
- [25] Rajat Khanda. 2024. Agentic AI-Driven Technical Troubleshooting for Enterprise Systems: A Novel Weighted Retrieval-Augmented Generation Paradigm. *arXiv preprint arXiv:2412.12006* (2024).
- [26] Mosh Levy, Alon Jacoby, and Yoav Goldberg. 2024. Same Task, More Tokens: the Impact of Input Length on the Reasoning Performance of Large Language Models. *arXiv:2402.14848* [cs.CL] <https://arxiv.org/abs/2402.14848>
- [27] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.
- [28] Yuan Li, Jun Hu, Jiaxin Jiang, Zemin Liu, Bryan Hooi, and Bingsheng He. 2025. RGL: A Graph-Centric, Modular Framework for Efficient Retrieval-Augmented Generation on Graphs. *arXiv preprint arXiv:2503.19314* (2025).
- [29] Zihao Li, Dongqi Fu, and Jingrui He. 2023. Everything evolves in personalized pagerank. In *Proceedings of the ACM Web Conference 2023*. 3342–3352.
- [30] IC Mogotsi. 2010. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze: Introduction to information retrieval: Cambridge University Press, Cambridge, England, 2008, 482 pp, ISBN: 978-0-521-86571-5.
- [31] Dhanachandra Ningthoujam, Shweta Yadav, Pushpak Bhattacharyya, and Asif Ekbal. 2019. Relation extraction between the clinical entities based on the shortest dependency path based LSTM. *arXiv preprint arXiv:1903.09941* (2019).
- [32] Rodrigo Nogueira and Kyunghyun Cho. 2020. Passage Re-ranking with BERT. *arXiv:1901.04085* [cs.IR] <https://arxiv.org/abs/1901.04085>
- [33] Farhad Nooralahzadeh, Lilja Øvrelid, and Jan Tore Lønning. 2018. Sirius-ltg-uio at semeval-2018 task 7: Convolutional neural networks with shortest dependency paths for semantic relation extraction and classification in scientific papers. *arXiv preprint arXiv:1804.08887* (2018).
- [34] Sarah Packowski, Inge Halilovic, Jenifer Schlotfeldt, and Trish Smith. 2024. Optimizing and Evaluating Enterprise Retrieval-Augmented Generation (RAG): A Content Design Perspective. In *Proceedings of the 2024 8th International Conference on Advances in Artificial Intelligence*. 162–167.
- [35] Boci Peng, Yun Zhu, Yongchao Liu, Xiaohu Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921* (2024).
- [36] Divyansh Singh, Manuel Nunez Martinez, Bonnie J. Dorr, and Sonja Schmer Galunder. 2025. SLIDE: Sliding Localized Information for Document Extraction. *arXiv:2503.17952* [cs.CL] <https://arxiv.org/abs/2503.17952>
- [37] Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391* (2024).
- [38] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports* 9, 1 (2019), 1–12.
- [39] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.
- [40] Zhiruo Wang, Jun Araki, Zhengbao Jiang, Md Rizwan Parvez, and Graham Neubig. 2023. Learning to Filter Context for Retrieval-Augmented Generation. *arXiv:2311.08377* [cs.CL] <https://arxiv.org/abs/2311.08377>
- [41] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of ‘small-world’networks. *nature* 393, 6684 (1998), 440–442.
- [42] Mingji Yang, Hanzhi Wang, Zhewei Wei, Sibo Wang, and Ji-Rong Wen. 2024. Efficient algorithms for personalized pagerank computation: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2024).
- [43] Michihiro Yasunaga, Antoine Bosselut, Hongyu Ren, Xikun Zhang, Christopher D Manning, Percy Liang, and Jure Leskovec. 2022. Deep Bidirectional Language-Knowledge Graph Pretraining. *arXiv:2210.09338* [cs.CL] <https://arxiv.org/abs/2210.09338>
- [44] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. 2019. Aligraph: A comprehensive graph neural network platform. *arXiv preprint arXiv:1902.08730* (2019).
- [45] Zulun Zhu, Siqiang Luo, Wenqing Lin, Sibo Wang, Dingheng Mo, and Chunbo Li. 2024. Personalized pageranks over dynamic graphs-the case for optimizing quality of service. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 409–422.