DELG: An Automatic Framework for Constructing Definition-Enriched Lightweight Domain Knowledge Graphs

Anonymous ACL submission

Abstract

Knowledge Graphs (KGs) are widely used to represent structured factual knowledge and serve as key resources for reasoning and question answering tasks. When combined with Large Language Models (LLMs), they enhance both interpretability and precision in complex reasoning tasks. However, integrating KGs with LLMs faces two critical challenges: (i) redundancy in triples, which leads to storage inefficiency and increased retrieval latency, and (ii) conceptual ambiguity caused by insufficient entity definitions, which limits reasoning accuracy.

To address these issues, we propose **DELG** (Definition Enriched Lightweight Graph), a framework designed to construct lightweight and semantically enriched knowledge graphs. DELG includes two key components. The *Redundancy Removing Component* identifies and eliminates both semantic and structural redundancies, thus reducing storage overhead while maintaining factual completeness and improving retrieval efficiency. The *Definition Enrich Component* hierarchically expands entity definitions based on entity complexity, ensuring that concept representations are contextually clear and semantically precise. We conduct extensive experiments on a medical dataset derived from *Treating Autoimmune Disease with Chinese Medicine* and evaluate DELG on LLM-based KG-QA tasks. Code and data are available at `https://anonymous.4open.science/r/DELG-FD53`.

1 Introduction

Knowledge Graphs (KGs) store large-scale structured factual data in form of triples, linking entities and their relationships (Heist et al., 2020; Xiong et al., 2017). KGs are widely used in applications such as search engines, social networks, and medical decision support, as they provide structured, queryable knowledge that is valuable across domains (Liu et al., 2020; Zhang et al., 2022). Recent advances in Large Language Models (LLMs) have made them central to tasks such as question answering, text generation, and language translation (Yang et al., 2024).

Recent research has increasingly explored the synergy between LLMs and KGs (Pan et al., 2024). The integration of KGs with LLMs has become a key area of research, especially in complex reasoning tasks. One prominent approach, knowledge graph retrieval-augmented generation (KG-RAG) (Sanmartin, 2024), uses KGs to retrieve relevant information that is then processed by an LLM to generate the final response. This approach combines the language understanding and reasoning ability of LLMs with the structured relational knowledge of KGs, enabling more accurate and contextually relevant answers in a variety of applications such as medical decision support, legal text retrieval, and financial compliance review (Hamza et al., 2025; Kalra et al., 2024; Huang et al.; Ma et al., 2021). Specifically, the process involves two main stages: in the construction phase, entities and relationships are extracted from raw database to build KGs; in the question-answering phase, a relevant subgraph is retrieved based on the user query, which is then fed into an LLM to generate the final answer. Compared to traditional KG-based QA methods, KG-RAG allows users to pose more diverse and flexible questions, providing more precise answers (Santos et al., 2022).

However, despite these potential benefits, there remain two major challenges in fully integrating KGs with LLMs. The first challenge is *redundancy in the KG*. When KGs are constructed from large raw corpora, they often contain duplicate or semantically overlapping triples that describe the same fact in different ways. For example, in a medical corpus, both triples *[“Rheumatoid arthritis” | is associated with | “Joint pain”]* and *[“Rheumatoid arthritis” | causes | “Joint discomfort”]* may appear, even though they convey nearly identi-

cal information. Such redundancy inflates storage costs and enlarges the search space during retrieval, which in turn increases latency in KG-RAG inference (Subagdja et al., 2024; Yang and Curry, 2024). The second challenge is *conceptual ambiguity*. Raw domain texts often lack explicit definitions for specialised or polysemous entities, leading to vague or underspecified concepts in the graph. For instance, in biomedical contexts, the entity *"Dampness"* may refer either to an environmental condition or to a specific pathogenic concept in traditional Chinese medicine. Without explicit definitional grounding, an LLM may misinterpret these entities, connect them to incorrect relations, or generate factually inconsistent answers (Zhang and Soh, 2024a; Keluskar et al., 2024).

To address these challenges, we propose **DELG** (Definition Enriched Lightweight Graph), a novel framework designed to alleviate both redundancy and conceptual ambiguity in knowledge graphs. For the first challenge, redundancy, DELG eliminates irrelevant and duplicate triples to reduce graph scale and storage overhead while improving retrieval efficiency and lowering query latency (Tan et al., 2019; Fu et al., 2019). For the second challenge, conceptual ambiguity, DELG introduces a hierarchical definition enrichment mechanism that selectively expands and refines entity definitions based on their complexity. This approach enables more precise and interpretable representations of domain concepts, thereby improving reasoning accuracy in KG-based question answering.

Example 1. As illustrated in Figure 1, the original corpus comes from a medical textbook in the field of Chinese medicine, which contains many domain-specific and easily confusable concepts. When facing the question *"Which pathogen originating from Dampness is described as thicker than Fluid?"*, a KG constructed by traditional methods fails to provide the correct answer due to insufficient conceptual definitions. Meanwhile, redundant structures in the KG not only occupy unnecessary storage space but also increase retrieval latency. In contrast, DELG effectively removes redundant triples and enriches entity definitions, enabling the LLM to reason more accurately based on a cleaner and more semantically expressive graph.

In this research, to minimize the influence of the LLMs' intrinsic knowledge on the evaluation results, we selected a publicly available medical textbook, *Treating Autoimmune Disease with Chinese Medicine*, as the raw text and applied our framework to generate the corresponding knowledge graph. We conducted extensive experiments on LLM-based KG-QA tasks using this graph. The results demonstrate that the knowledge graph constructed with our approach is smaller in scale, more efficient in retrieval, and yields more accurate answers. In addition, our experiments also explore the potential of applying DELG in locally deployed large language model environments.

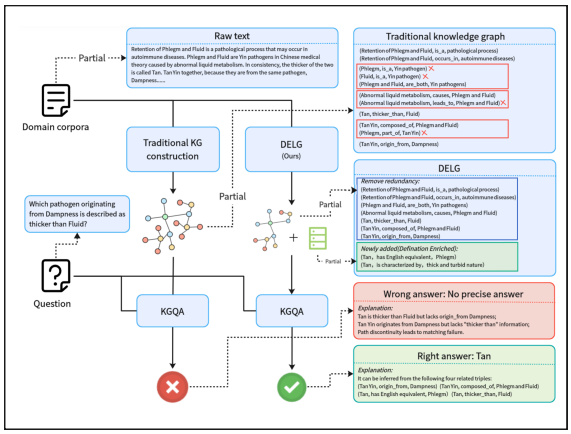

Figure 1: Comparison between DELG and traditional KG construction methods. In the KG-QA scenario shown, DELG successfully produces the correct answer by reducing redundancy and enriching entity definitions.

In summary, the main contributions of this paper are as follows:

- We propose an automatic knowledge graph construction framework that performs both semantic and structural redundancy elimination, thereby improving storage efficiency and retrieval performance in KG-based QA tasks (Sections 3.1–3.2).

- We introduce a novel method for entity definition enhancement, which computes the complexity of entities and selectively expands their definitions. This approach aims to reduce ambiguity and improve the accuracy of LLM-based QA systems (Section 3.3).

- We propose a method for the automatic generation of QA pairs from raw corpus data. Using this method, we generate 100 QA pairs and evaluate the performance of LLM-based QA systems, demonstrating its effectiveness in improving answer accuracy and retrieval efficiency (Section 4).

2 Preliminaries

We first introduce two key concepts that form the foundation of our approach: **Knowledge Graph Redundancy Types** and **Entity Complexity**.The formal definitions are provided in Appendix B.

Knowledge Graph Redundancy: Knowledge graph redundancy can be classified into two major types: "semantic redundancy" and "structural redundancy". Semantic redundancy occurs when two triples in the graph have the same or nearly identical meanings. This happens when their head entity, relation, and tail entity are semantically equivalent, which leads to unnecessary duplication of information in the graph. Structural redundancy refers to situations where one triple can be derived from another, based on the logical relationships within the graph. Addressing these redundancies is essential for optimizing both storage and retrieval efficiency in KGs (Hu and Wang, 2025; Wu et al., 2014).

Entity Complexity: Entity complexity refers to the intricacy or specialization of an entity in the knowledge graph. We borrow from Shannon's information theory, where the complexity of an entity is inversely related to its frequency of occurrence in the corpus. An entity that appears infrequently carries more information and is therefore considered more complex. Additionally, the specialization of an entity is considered: entities that frequently appear in specialized domains, such as medical or legal corpora, but rarely in general-purpose corpora, are regarded as more complex due to their specialized knowledge. This makes specialized entities harder to define and more difficult to handle in KGs (Shannon, 1948; Ahmad et al., 1999).

3 Methodology

In this section, we present DELG, a framework designed to address redundancy and entity definition enrichment in KGs. DELG consists of three components: **Generate Component**, which extracts triples from domain-specific corpora to form an initial knowledge graph (Section 3.1); **Redundancy Removing Component**, which eliminates semantic and structural redundancy to create a lightweight graph (Section 3.2); **Definition Enrich Component**, which refines entity definitions based on complexity to reduce ambiguity (Section 3.3). The overall architecture of DELG is illustrated in Figure 2. A theoretical analysis of the time complexity for these components is provided in Appendix C.

3.1 Generate Component

The Generate Component is responsible for extracting triples from domain-specific corpora to form the initial knowledge graph. This process simulates the real-world extraction of structured knowledge from raw textual data and can be replaced with other triple extraction methods as needed.

To handle long documents efficiently, the input text is first split into chunks of a predefined maximum length. Each chunk is then processed to resolve coreferences, ensuring that pronouns and other referring expressions are replaced with the appropriate noun phrases. This step improves the accuracy of subsequent triple extraction.

Triples are extracted from each resolved chunk using a large language model according to specific constraints: entities must appear verbatim in the text and consist of no more than three words, predicates must correspond to expressions in the text, and vague or speculative triples are omitted. The batch size of chunks can influence the number of triples generated per batch, affecting the density and coverage of the resulting knowledge graph. All triples from the chunks are then aggregated to form a preliminary, **domain-specific knowledge graph**. In this work, the target domain is *Chinese Medicine*, and the raw text corpus used for extraction is the book *Treating Autoimmune Disease with Chinese Medicine*. For a comparison of our method with the spaCy-based method, see Appendix D.1

3.2 Redundancy Removing Component

After generating the initial domain-specific KG, the next step is to remove redundant triples to obtain a lightweight, more efficient graph. This component addresses all types of redundancy discussed in Section 2: **Semantic Redundancy (SR)**, **Bidirectional Redundancy (BR)**, **Transitive Redundancy (TR)**, and **Functional Dependency Redundancy (FDR)**. Each type is handled by a dedicated procedure, and the overall process refines the graph while preserving essential knowledge. For a comparison of our method with rule-based redundancy removal method, see Appendix D.2.

3.2.1 Semantic Redundancy Removal (SR)

Semantic redundancy arises when multiple entities or relations are semantically equivalent. To detect SR, entity and relation embeddings are first obtained using an LLM. Cosine similarity is computed between embeddings, and items exceeding a similarity threshold are clustered. For each cluster,

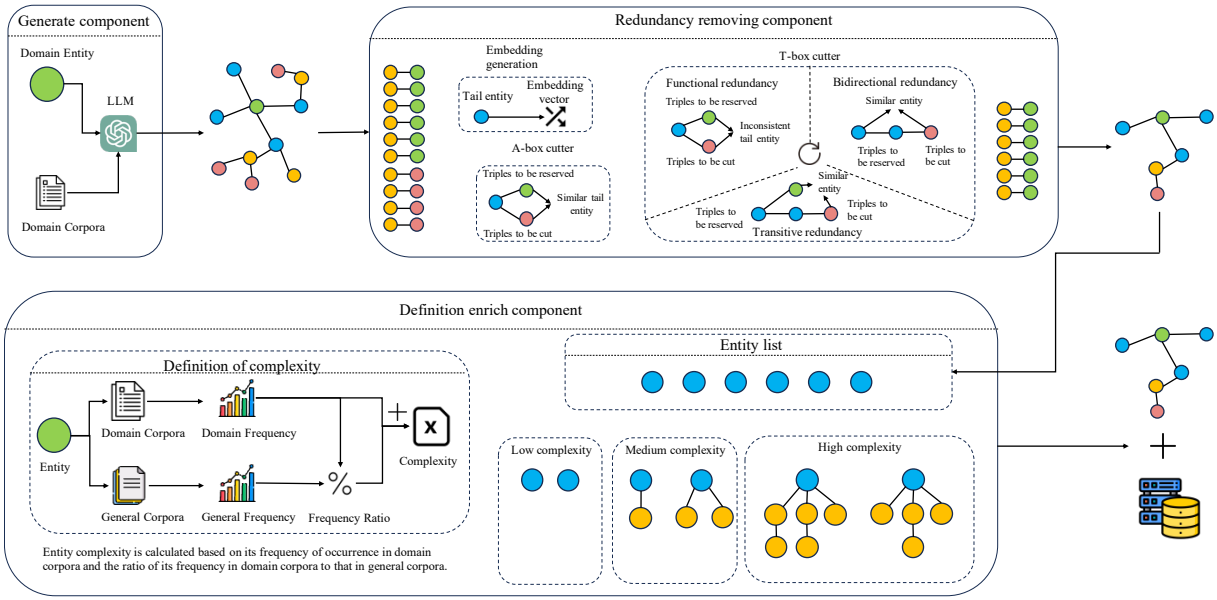

Figure 2: The overview of our proposed DELG. The *Generate Component* converts the raw corpus into an initial knowledge graph. The *Redundancy Removing Component* eliminates both semantic and structural redundancies. The *Definition Enrich Component* expands entity information based on complexity, yielding a definition-enriched lightweight knowledge graph.

a representative entity or relation is chosen based on frequency, and all occurrences in the triples are replaced accordingly. Finally, exact duplicate triples (h, r, t) are removed. This process ensures that semantically similar concepts are unified while maintaining the integrity of the graph.

3.2.2 Functional Dependency Redundancy Removal (FDR)

FDR targets triples where a relation is inherently functional, i.e., each head entity corresponds to a unique tail entity. For each (h, r) pair, if multiple tails exist, an LLM is queried to verify whether the relation is functional. If confirmed, only one representative triple is kept; otherwise, all triples are retained. This step eliminates redundant triples that violate functional constraints while preserving necessary diversity for non-functional relations.

3.2.3 Bidirectional Redundancy Removal (BR)

BR occurs when a triple and its inverse represent the same information, e.g., (h, r, t) and (t, r^{-1}, h). To remove BR, each triple is normalized by sorting the head and tail entities, and duplicates under this normalization are removed. Self-loops $(h = t)$ are retained. This ensures that redundant inverses do not consume additional storage or introduce potential inconsistencies.

3.2.4 Transitive Redundancy Removal (TR)

TR is present when one triple can be logically inferred from others, such as (h_1, r_1, t_1) and (t_1, r_2, t_2) implying (h_1, r_3, t_2). To detect TR, candidate triples are checked against inferred triples using a combination of graph traversal and LLM reasoning. Triples confirmed to be derivable are removed from the graph. This step reduces unnecessary repetition while preserving the knowledge needed for reasoning tasks.

Through the sequential application of SR, FDR, BR, and TR removal, the knowledge graph is transformed into a lightweight version, retaining critical domain-specific knowledge while improving storage and retrieval efficiency.

3.3 Definition Enrich Component

In this section, we first define entity complexity on information-theoretic and domain-specific grounds, then use this signal to drive a hierarchical definition enrichment procedure, which yields clearer definitions for difficult concepts and keeps the graph compact. The overall outcome is a definition-enriched yet lightweight domain graph.

3.3.1 Entity Complexity Evaluation

The complexity of an entity is determined by two complementary measures defined in Section 3: the *single-document frequency* $I(w)$ and the *specialization ratio* $W(w)$. Since our raw corpus itself is

domain-specific, these two measures incorporate the same frequency statistics but in different ways. For the first measure, the frequency of an entity in the given domain corpus is directly used to compute its information amount, reflecting how much information the entity carries within the local context. For the second measure, the same frequency is used as the domain-side count and is compared against the frequency of the entity in a general-purpose corpus, thereby capturing how specialized the entity is relative to general language usage. In both cases, the values are normalized by logarithmic scaling and bounded within fixed ranges, as explained in Section 2. The final entity complexity is defined as the sum of these two normalized values, which ensures that the overall score lies within a stable interval. This formulation allows the framework to identify entities that are both informative and domain-specific, highlighting them as complex entities that require enriched definitions.

3.3.2 Recursive Definition Expansion

The procedure takes as input the set of entities that appear in the current knowledge graph, collected from all triples (h, r, t). For each entity e, the system evaluates C_e using Section 3. If C_e is below the complexity threshold τ, expansion stops for e. If $C_e \geq \tau$, a definition-oriented RAG module proposes candidate definitional triples of the form (e, r, t). The system retains only candidates that satisfy a monotone constraint on complexity, namely the tail entity must be strictly simpler than the head, $C_t < C_e$. This gives a local explanation of a complex entity by several simpler entities. If a retained tail still has $C_t \geq \tau$, the algorithm recurses on that tail until either the depth limit is reached or the entity becomes simple.

To optimize expansion and avoid redundancy, the system uses a global index that maps each unique entity string to a canonical identifier, storing edges as $[r, \mathrm{id}(t)]$. This index prevents duplicate storage and allows constant-time checks for previously added definition edges. The system also employs a per-entity top M cap and a maximum recursion depth to limit growth. A thread-safe design with a shared index and locks supports parallel expansion over candidate heads. This ensures a balanced expansion: complex entities receive richer definitional neighborhoods, while simple entities expand minimally, reducing ambiguity where needed and preserving a lightweight graph elsewhere.

4 Experiments

To evaluate the effectiveness of our framework, we conduct comprehensive experiments in an LLM-based KG-QA scenario. The experimental setup is presented in Section 4.1. We design an automated method for generating 100 question–answer pairs from the constructed knowledge graph (Section 4.2). The main experiments examine the effects of redundancy removal, definition enrichment, and model size on KG-QA performance, focusing on recall and accuracy (Section 4.3). We then analyze the influence of both components on inference latency (Section 4.4) and further study the impact of different parameter settings on the quality of graph construction and downstream QA tasks (Section 4.5). Finally, we investigate the performance of the framework in general domains, validating its generalization ability (Section 4.6). For further experimental analysis, see Appendix H.

4.1 Experimental Setup

To minimize the influence of the LLM's inherent knowledge, we select the book *Treating Autoimmune Disease with Chinese Medicine* as the domain-specific data source. The corpus contains approximately 756K tokens. After preprocessing and the initial generation described in Section 3.1, the resulting knowledge graph includes 9,350 triples and 4,160 distinct entities. We store the graph in Neo4j to simulate a realistic KG-QA environment. For each input question, we retrieve a subgraph using vector alignment and breadth-first search (BFS), and feed it as a prompt into the LLM to generate an answer. The retrieval process is detailed in Appendix E. The recall rate measures the proportion of expected entities covered by the retrieved subgraph:

$$\text{Recall} = \frac{|\text{Entities}_{\text{retrieved}} \cap \text{Entities}_{\text{expected}}|}{|\text{Entities}_{\text{expected}}|}. \quad (1)$$

Accuracy reflects the semantic correctness of generated answers, scored automatically by GPT-4o based on reference answers.

The framework consists of two components. The Redundancy Removing Component uses a similarity threshold (set to 0.7 in the main experiments of Section 4.3) to merge semantically redundant entities, with its effect on accuracy and efficiency analyzed in Section 4.5. The Definition Enrich Component determines each entity's expansion depth using a complexity threshold of 0.65.

Table 1: Comprehensive evaluation results of recall and accuracy across different QA types, KG configurations, and LLM backbones.

	GPT-4o-mini						Qwen2.5-7B-Instruct					
	Recall			Accuracy			Recall			Accuracy		
Configuration	Struct.	Def.	Complex	Struct.	Def.	Complex	Struct.	Def.	Complex	Struct.	Def.	Complex
Full KG	0.61	0.50	0.44	0.48	0.20	0.03	**0.61**	0.43	0.53	**0.21**	0.25	0.05
Red.-KG	0.65	0.42	0.40	0.36	0.13	0.16	0.49	0.39	0.42	0.12	0.18	0.07
Red.-KG + is_A Def. (Roller et al., 2018)	0.67	0.53	0.47	0.36	0.30	0.15	0.57	0.47	0.47	0.17	0.28	**0.25**
Red.-KG + Rec. Def. (Ours)	**0.72**	**0.68**	**0.50**	**0.49**	**0.62**	**0.18**	0.52	**0.51**	**0.55**	0.18	**0.29**	0.16

Struct.: structure-only QA, testing relational reasoning; **Def.**: definition-only QA, testing entity understanding; **Complex**: merging relational and definitional reasoning into a complex path; **Recall** measures the coverage of expected entities in the retrieved subgraph; **Accuracy** denotes the correctness of the final answer scored by *GPT-4o* against ground-truth references. **Red.-KG**: redundancy-reduced graph; **is_A Def.**: simple "is A" enrichment; **Rec. Def.**: recursive hierarchical enrichment (ours).

We also compare this approach with a traditional *is_A*-based enrichment method, which stores explicit definitions and indexes for each entity (see Appendix F) (Subagdja et al., 2024). To evaluate performance across models of different parameter scales, we employ two representative LLMs: *GPT-4o-mini* (cloud-based) and *Qwen-2.5-7B-Instruct* (locally deployed).

4.2 Automatic QA Generation Method

To facilitate consistent evaluation, we design a fully automated method that generates QA pairs directly from the knowledge graph, specifically for validating the effectiveness of our approach in LLM-based KG-QA tasks. Three distinct QA categories are defined: (i) structure-only QA pairs that rely solely on in-graph structural relations, (ii) definition-only QA pairs focused on definitional triples, and (iii) Complex QA pairs that combine both structural and definitional information.

(i) Structure-only QA pairs. The Method first filters the raw triples. For each triple, LLM infers the tail entity from the given head and relation. The triple is retained only if the embedding similarity between the generated and original tails exceeds 0.5, ensuring that the resulting reasoning chains are well-posed and lead to unique, valid answers. From the filtered graph, we construct multi-hop chains of two to three intermediate relations as the basis for QA generation. The head entity and intermediate relations are presented to the LLM, which generates a question asking what entity is reached after sequentially applying these relations. The correct answer is the chain's tail entity. For instance, a two-hop path [E1, R1, E2, R2, E3] produces a question involving E1, R1, and R2, and the expected answer is E3.

(ii) Definition-only QA pairs. To construct definition-based questions, we first enhance the KG using the RAG-based Definition Enrich Component, generating definition triples in the form [entity, relation, definition]. For example, ["ALT level", "measures", "aminotransferase"]. These triples are concatenated into reasoning chains similar to the structure-only case, except that auxiliary relations such as copulas are removed to maintain semantic clarity. The LLM is then prompted to produce natural questions where the answer depends on the definitional relation between entities.

(iii) Complex QA pairs. Complex QA pairs combine the previous two types, integrating both structural and definitional elements. Specifically, two or three structural triples are combined with one or two definitional triples, forming a mixed chain. These QA pairs represent a reasoning path that begins with a concrete entity, traverses through multiple relational hops to reach an abstract or complex concept, and finally connects via a definitional edge to generate an interpretable description. This type of QA pair evaluates both the system's reasoning capacity and its ability to produce meaningful, interpretable answers.

We automatically construct 100 QA pairs for evaluation, including 40 structure-only, 40 definition-only, and 20 Complex questions. Detailed prompting examples and generation templates are provided in Appendix G.

4.3 Comprehensive Evaluation

The experimental results are presented in Table 1. The table reports recall and accuracy across three QA types—Structure-only QA, Definition-only QA, and Complex QA—under six configurations of the knowledge graph, using two large language models: *GPT-4o-mini* and *Qwen-2.5-7B*. Here, "Full KG" refers to the knowledge graph that has not been pruned for redundancy. "is_A Def." indicates the traditional "is A" definition enrichment.

"Rec. Def." denotes our proposed recursive definition enrichment method. "Red.-KG" refers to the knowledge graph after redundancy removal (Section 3.2). Each metric is averaged over the results of all QA pairs of the corresponding type.

For the *GPT-4o-mini* model, redundancy removal leads to an average recall decrease of less than 10%, with accuracy remaining stable or even improving in some cases. This suggests that even though the redundancy-reduced graph contains fewer triples, it does not necessarily result in lower accuracy. In fact, the reduction in redundant information from the graph likely reduces noise, thereby enhancing the model's reasoning accuracy in certain instances. The definition enrichment module significantly improves recall. Compared to the "is_A" extension, our recursive enrichment method leads to notable gains in both recall and accuracy, with at least a 15 percentage point improvement in Definition-only QA pairs. However, *GPT-4o-mini* shows lower accuracy in Complex QA pairs, suggesting that the model's reasoning capability still struggles with more complex, multi-hop questions.

For the *Qwen-2.5-7B* model, the overall recall is approximately 50%, with answer accuracy around 20%. The application of redundancy removal results in only a slight decrease in both recall and accuracy, confirming that core facts remain intact. The definition enrichment module leads to modest improvements in recall and accuracy, with gains similar to those from the *is_A* definition enhancement method. This suggests that *Qwen-2.5-7B* has limited reasoning capacity, whether for structure-only, definition-only, or Complex questions.

4.4 Inference Latency Analysis

We tested the impact of both components on the time required for QA tasks, dividing the process into retrieval and answer phases. Specifically, we evaluated the inference latency for 100 QA pairs across the four configurations shown in Table 2 and calculated the average latency.

As shown in the table, the answer phase latency is an order of magnitude smaller than the retrieval phase latency, with minimal variation across different configurations. In the retrieval phase, the redundancy removal module reduced the inference latency by approximately 2 seconds, or about 10%, consistent with the theoretical time complexity analysis presented in Appendix C. However, the definition enrichment module did not lead to increased inference latency. On the contrary, it significantly reduced the retrieval latency, exceeding our theoretical expectations from Appendix C. This improvement is likely due to the additional definition information making concepts clearer for the LLM, thus reducing the selection of irrelevant nodes during subgraph retrieval, ultimately enhancing retrieval efficiency.

Table 2: Average Latency Comparison of Different KG Configurations Using GPT-4o-mini

Configuration	Retrieval (s)	Answer (s)
Full KG	21.39	1.74
Full KG + Rec. Def.	18.40	1.73
Red.-KG	19.94	1.66
Red.-KG + Rec. Def. (Ours)	15.88	1.64

Retrieval latency: average time to retrieve the subgraph; **Answer latency**: average time for LLM to generate the result; **Red.-KG**: redundancy-reduced KG; **Rec.Def**: recursive definition enrichment.

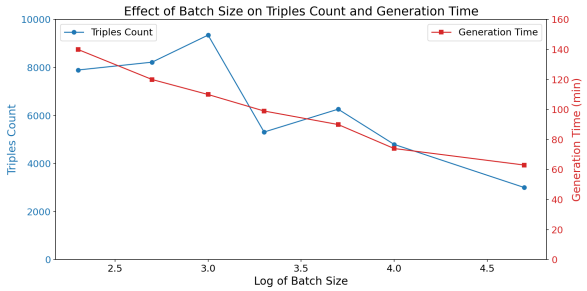

Figure 3: Effect of batch size on triples count and generation time. The x-axis represents the logarithm (base 10) of the batch size values, ranging from 200 to 50,000.

4.5 Effect of Parameter Settings

4.5.1 Effect of Chunk Size

As mentioned in Section 3.1, the batch size of chunks can influence the number of triples generated per batch, affecting both the density and coverage of the resulting knowledge graph. In this subsection, we test different batch sizes ranging from 200 to 50,000, comparing their performance in terms of construction time and the number of generated triples.

As shown in Figure 3, the time taken for knowledge graph construction decreases gradually with increasing batch size, though the reduction is not substantial. This is because, although a larger batch size results in fewer API calls, each individual API call involves more tokens. The number of triples initially increases and then decreases as the batch size grows. The increase occurs because larger batch sizes allow the LLM to capture relationships

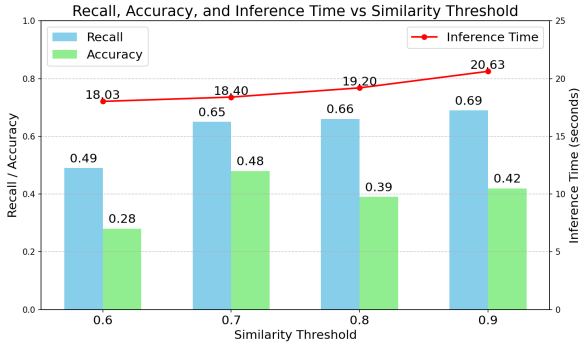

Figure 4: Comparison of recall, accuracy, and inference latency across different similarity thresholds.

Table 3: Average Latency Comparison of Different KG Configurations Using GPT-4o-mini

Configuration	Recall	Accuracy
Full KG	0.74	0.51
Red.-KG + Rec. Def. (Ours)	0.77	0.64

between entities that are further apart. However, the decrease is due to the fact that excessively long texts can dilute the attention of the LLM, making it less capable of extracting all relevant triples. In summary, the best graph construction performance is achieved when the batch size is set to 1,000.

4.5.2 Effect of Redundancy Threshold

As shown in Figure 4, we test the performance across different similarity thresholds. To ensure fairness, we test the *GPT-4o-mini* model on 100 QA pairs at each threshold in the same environment.

We can observe that as the similarity threshold decreases from 0.9 to 0.6, the inference latency gradually decreases. However, from 0.7 to 0.6, there is a noticeable drop in both recall and accuracy, indicating that at a threshold of 0.6, redundancy removal significantly affects the retention of factual information. The highest accuracy is achieved at a threshold of 0.7, suggesting that a certain level of redundancy removal reduces interference from the LLM, thereby improving accuracy. Overall, it can be seen that a threshold of 0.7 appears to be a reasonable choice in this experiment. In real-world applications, users can adjust the threshold according to their specific needs to balance inference latency and response accuracy.

4.6 Generalization Discussion

To demonstrate the generalization ability of our method, we first collected a corpus from 28 different domains, resulting in approximately 1 million tokens, which is slightly larger than the corpus used in the main experiment, *Treating Autoimmune Disease with Chinese Medicine*. We then applied our framework to construct the corresponding lightweight, definition-enriched knowledge graphs and generated 100 question-answer pairs.

As shown in Table 3, our method achieves improvements in recall and accuracy at approximately 80% of the graph size compared to the original graph. It can be observed that, for QA tasks based on the general-domain knowledge graph, both recall and accuracy are relatively high, regardless of whether our method is applied. This is because general-domain knowledge graphs tend to have fewer specialized entities, making it less likely for the LLM to confuse them. While the advantage of our method is more pronounced in specialized domains, it still helps reduce storage space in general domains.

5 Related Work

Recent studies explore LLM-based automatic knowledge graph construction and retrieval-augmented generation with KG-enhanced reasoning to improve factual accuracy and interpretability; however, existing methods often suffer from redundant graph structures and semantic ambiguity caused by insufficient entity definitions, which limits its reasoning performance, motivating our proposal of DELG for lightweight and definition-enriched knowledge graph construction. We give a more detailed description in Appendix A

6 Conclusion

In this paper, we propose **DELG**, an automated framework specifically designed to tackle the critical challenges of redundancy and conceptual ambiguity in the integration of KGs with LLMs. DELG introduces two main components. The redundancy removal process eliminates irrelevant and duplicate triples, improving retrieval efficiency, while the definition enrichment component refines entity definitions, ensuring that the knowledge graph captures more precise and contextually clear concepts. Our experiments show that DELG enhances answer accuracy and retrieval efficiency. In the future, we plan to validate the scalability of DELG in more domains and further optimize the method to improve its effectiveness in various practical applications.

Limitations

While DELG demonstrates excellent performance in domain-specific knowledge graph question answering (KG-QA), there are several limitations to our approach. First, our evaluation is limited to the KG-QA downstream task, and further experiments across other dimensions are needed. Second, our method has not been tested on larger-scale knowledge graphs, and future work should involve the construction, redundancy removal, and definition enrichment of graphs at the 1 billion scale. Finally, although our automated QA generation approach considers various scenarios, the quality of generated QA pairs cannot be fully guaranteed, and human-assisted validation remains necessary. Future efforts will focus on finer-grained scenario classification and rule design to improve QA generation quality.

References

Khurshid Ahmad, Lee Gillam, Lena Tostevin, and 1 others. 1999. University of surrey participation in trec8: Weirdness indexing for logical document extrapolation and retrieval (wilder). In *TREC*, pages 1–8.

Vahan Arsenyan, Spartak Bughdaryan, and 1 others. 2023. Large language models for biomedical knowledge graph construction: Information extraction from emr notes. In *Workshop on BNLP*.

Farah Atif, Ola El Khatib, and Djellel Difallah. 2023. Beamqa: Multi-hop knowledge graph question answering with sequence-to-sequence prediction and beam search. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 781–790.

Caio Viktor S Avila, Marco A Casanova, and 1 others. 2024. A framework for question answering on knowledge graphs using large language models. In *ICSC*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, and 1 others. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Johannes Frey, Lars-Peter Meyer, and 1 others. 2024. Assessing the evolution of llm capabilities for knowledge graph engineering in 2023. In *ESWC*.

J. Fu, Q. Li, and L. Ma. 2019. Reducing redundancy in knowledge graphs. *Knowledge-Based Systems*.

Xinyan Guan, Yanjiang Liu, and 1 others. 2024. Mitigating large language model hallucinations via autonomous knowledge graph-based retrofitting. In *AAAI*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Ameer Hamza, Yong Hyun Ahn, Sungyoung Lee, Seong Tae Kim, and 1 others. 2025. Llava needs more knowledge: Retrieval augmented natural language generation with knowledge graph for explaining thoracic pathologies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 3311–3319.

Nicolas Heist, Sven Hertling, Daniel Ringler, and Heiko Paulheim. 2020. Knowledge graphs on the web–an overview. *Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges*, pages 3–22.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, Adriane Boyd, and 1 others. 2020. spacy: Industrial-strength natural language processing in python.

Songhua Hu and Hengxin Wang. 2025. Del: a strategy for resolving redundancy in entity pairs within dual entity linker for relational triple extraction. *Evolutionary Intelligence*, 18(1):28.

Zihao Huang, Kelvin Du, Xulang Zhang, Rui Mao, and Erik Cambria. Combining llm-generated knowledge graphs with rag for financial sentiment extraction.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. *arXiv preprint arXiv:2212.00959*.

Rishi Kalra, Zekun Wu, Ayesha Gulley, Airlie Hilliard, Xin Guan, Adriano Koshiyama, and Philip Treleaven. 2024. Hypa-rag: A hybrid parameter adaptive retrieval-augmented generation system for ai legal and policy applications. *arXiv preprint arXiv:2409.09046*.

Aryan Keluskar, Amrita Bhattacharjee, and Huan Liu. 2024. Do llms understand ambiguity in text? a case study in open-world question answering. In *2024 IEEE International Conference on Big Data (BigData)*, pages 7485–7490. IEEE.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019a. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019b. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*.

Hanieh Khorashadizadeh, Fatima Zahra Amara, Morteza Ezzabady, Frédéric Ieng, Sanju Tiwari, Nandana Mihindukulasooriya, Jinghua Groppe, Soror Sahri, Farah Benamara, and Sven Groppe. 2024. Research trends for the interplay between large language models and knowledge graphs. *arXiv preprint arXiv:2406.08223*.

Yading Li, Dandan Song, and 1 others. 2024. A framework of knowledge graph-enhanced large language model based on question decomposition and atomic retrieval. In *EMNLP*.

Ying Lin, Heng Ji, Fei Huang, and Lingfei Wu. 2020. A joint neural model for information extraction with global features. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 7999–8009.

Q. Liu, X. Ren, and H. Zhang. 2020. Graph-based methods for redundancy removal in knowledge graph construction. *Journal of Artificial Intelligence Research*.

Robert L Logan IV, Nelson F Liu, Matthew E Peters, Matt Gardner, and Sameer Singh. 2019. Barack's wife hillary: Using knowledge-graphs for fact-aware language modeling. *arXiv preprint arXiv:1906.07241*.

Haoran Luo, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, Yifan Zhu, and 1 others. 2023a. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. *arXiv preprint arXiv:2310.08975*.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2023b. Reasoning on graphs: Faithful and interpretable large language model reasoning. *arXiv preprint arXiv:2310.01061*.

H. Ma, X. Ren, and H. Zhang. 2021. Efficient knowledge graph construction using llms. *ACM Computing Surveys*.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

R OpenAI. 2023. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2(5):1.

X. Pan, L. Ma, and J. Fu. 2024. Unifying large language models and knowledge graphs for complex question answering. *IEEE Transactions on Neural Networks and Learning Systems*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Stephen Roller, Douwe Kiela, and Maximilian Nickel. 2018. Hearst patterns revisited: Automatic hypernym detection from large text corpora. *arXiv preprint arXiv:1806.03191*.

Diego Sanmartin. 2024. Kg-rag: Bridging the gap between knowledge and creativity. *arXiv preprint arXiv:2405.12035*.

A. Santos, A. Lima, and C. Ferreira. 2022. Knowledge graphs in healthcare: Applications and advances. *Journal of Healthcare Informatics Research*.

CE Shannon. 1948. A mathematical theoryof communication, bell system tech. *J*, 27:379–423.

Budhitama Subagdja, D Shanthoshigaa, Zhaoxia Wang, and Ah-Hwee Tan. 2024. Machine learning for refining knowledge graphs: A survey. *ACM Computing Surveys*, 56(6):1–38.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M Ni, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. *arXiv preprint arXiv:2307.07697*.

J. Tan, X. Zhang, and L. Ma. 2019. Redundancy in knowledge graphs: Approaches and challenges. *Journal of Knowledge Management*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Somin Wadhwa, Silvio Amir, and Byron C Wallace. 2023. Revisiting relation extraction in the era of large language models. In *Proceedings of the conference. association for computational linguistics. meeting*, volume 2023, page 15566.

Honghan Wu, Boris Villazon-Terrazas, Jeff Z Pan, and Jose Manuel Gomez-Perez. 2014. How redundant is it?: An empirical analysis on linked datasets. In *Proceedings of the 5th International Workshop on Consuming Linked Data (COLD 2014) co-located with the 13th International Semantic Web Conference (ISWC 2014) Riva del Garda, Italy, October 20, 2014*. CEUR-WS.

Chenyan Xiong, Russell Power, and Jamie Callan. 2017. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the 26th international conference on world wide web*, pages 1271–1279.

Tianhan Xu, Yixun Gu, and 1 others. 2024a. Knowledge graph construction for heart failure using large language models with prompt engineering. *Frontiers in Computational Neuroscience*.

Yao Xu, Shizhu He, Jiabei Chen, Zihao Wang, Yangqiu Song, Hanghang Tong, Guang Liu, Kang Liu, and Jun Zhao. 2024b. Generate-on-graph: Treat llm as both agent and kg in incomplete knowledge graph question answering. *arXiv preprint arXiv:2404.14741*.

Yang Yang and Edward Curry. 2024. Open knowledge base canonicalization: Techniques and challenges.

Z. Yang, X. Pan, and T. Zhao. 2024. Facts and reasoning with knowledge graphs and large language models. *ACM Transactions on Knowledge Discovery from Data*.

Jason Youn and Ilias Tagkopoulos. 2022. Kglm: Integrating knowledge graph structure in language models for link prediction. *arXiv preprint arXiv:2211.02744*.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Wang, Zhiguo Wang, and Bing Xiang. 2022. Decaf: Joint decoding of answers and logical forms for question answering over knowledge bases. *arXiv preprint arXiv:2210.00063*.

H. Yu, J. Wei, and L. Zhao. 2020. Enhancing knowledge graph accuracy in question answering systems. *Journal of Machine Learning Research*.

Bowen Zhang and Harold Soh. 2024a. Extract, define, canonicalize: An llm-based framework for knowledge graph construction. *arXiv preprint arXiv:2404.03868*.

Bowen Zhang and Harold Soh. 2024b. Extract, define, canonicalize: An llm-based framework for knowledge graph construction. In *EMNLP*.

J. Zhang, Z. Yang, and X. Tan. 2022. Ontoprotein: A knowledge graph for protein research. *Bioinformatics*.

Yun Zhu, Yaoke Wang, Haizhou Shi, and Siliang Tang. 2024. Efficient tuning and inference for large language models on textual graphs. *arXiv preprint arXiv:2401.15569*.

A Related Work

A.1 LLM-Based Automatic Knowledge Graph Construction

Recent advances show that large language models (LLMs) can automate the construction of knowledge graphs (KGs) by directly extracting entities and relations from unstructured text, thereby significantly reducing the need for manual annotation (Zhang and Soh, 2024b). With predefined or induced schemas, these models convert documents into structured triples and maintain high recall even in specialised domains such as biomedicine and healthcare (Xu et al., 2024a; Arsenyan et al., 2023). Frameworks such as Auto-KGQA and KELDaR integrate extraction, refinement, and verification into unified pipelines, enabling near–real-time updates while mitigating factual hallucination (Avila et al., 2024; Li et al., 2024). In addition, retrieval-based cross-checking mechanisms and benchmark-driven evaluations further enhance the consistency of automatically generated KGs (Guan et al., 2024; Frey et al., 2024). Despite these advances, several issues remain unresolved: hallucination persists when input evidence is sparse or noisy, and most pipelines still lack explicit conceptual explanations, which limits interpretability in reasoning tasks (Yu et al., 2020). Our proposed framework, DELG, extends this line of work by not only reducing redundant triples but also enriching entity definitions in a hierarchical manner to address conceptual ambiguity.

A.2 Retrieval-Augmented Generation and KG-Enhanced QA

Retrieval-Augmented Generation (RAG) combines parametric language models with non-parametric memory to improve factual accuracy and interpretability (Khandelwal et al., 2019a; Guu et al., 2020). Early approaches such as REALM and RAG jointly trained dense retrievers and generators, while later works like kNN-LM and RETRO scaled retrieval to trillion-token corpora (Khandelwal et al., 2019b; Borgeaud et al., 2022). Traditional text-centric RAG, however, overlooks relational structures crucial for multi-hop reasoning. GraphRAG addresses this by retrieving subgraphs instead of raw text, allowing LLMs to leverage graph topology for enhanced contextual reasoning (Zhu et al., 2024). Building on this, KG-enhanced RAG explicitly integrates entity–relation triples into retrieval. Models such as KGLM demonstrate that including KG-based facts helps LLMs better handle out-of-vocabulary entities and domain-specific relations (Logan IV et al., 2019; Youn and Tagkopoulos, 2022). Nonetheless, incomplete or ambiguous KGs can still lead to factual errors during inference. DELG mitigates this limitation by constructing lightweight, redundancy-reduced, and definition-enriched KGs that serve as higher-quality retrieval sources for LLM-based reasoning.

A.3 Complex Reasoning over Knowledge Graphs

Complex reasoning over knowledge graphs aims to answer multi-hop questions by performing structured reasoning on interconnected entities (Atif et al., 2023; Khorashadizadeh et al., 2024; Luo et al., 2023a). Existing approaches can be divided into semantic-parsing and retrieval-augmented methods. Semantic-parsing methods translate questions into executable formal queries such as SPARQL (Yu et al., 2022). Recent studies employ large-scale pre-trained models (e.g., T5 or LLaMA) to generate query expressions directly, improving the accuracy of formal query generation (Raffel et al., 2020; Touvron et al., 2023). In contrast, retrieval-augmented approaches (Jiang et al., 2023, 2022; Luo et al., 2023b) collect relevant triples and feed them to LLMs for reasoning. Advanced systems like ToG (Sun et al., 2023) and GoG (Xu et al., 2024b) further combine iterative exploration and graph traversal, achieving state-of-the-art performance. However, most existing methods rely on closed-source APIs (e.g., GPT-4 (OpenAI, 2023)), which makes them less effective when applied to locally deployed, smaller models. In this context, our DELG framework offers a complementary perspective: by reducing redundancy and enriching definitions, it improves reasoning accuracy and retrieval efficiency, and explores the potential of localized deployment of language models.

B Formal Definition

Based on the previously discussed base concepts, we now formalize the definitions for the two key metrics: redundancy and entity complexity.

Semantic Redundancy (SR): Given two entities, we calculate their semantic similarity by obtaining embeddings for each entity using a large language model (LLM). The cosine similarity between these embeddings is then computed. If the similarity score exceeds a predefined threshold, the two entities are considered semantically redundant. This can be expressed as follows.

$$\text{similarity}(e_1, e_2) = \frac{e_1 \cdot e_2}{\|e_1\|\|e_2\|} \tag{2}$$

where e_1 and e_2 represent the embeddings of entities e_1 and e_2, respectively, and $\|\cdot\|$ denotes the vector norm. If $\text{similarity}(e_1, e_2) \geq \theta$, where θ is the similarity threshold, then e_1 and e_2 are considered semantically redundant.

Bidirectional Redundancy (BR): Consider two triples in the knowledge graph:

$$T_1 = (h_1, r, t_1) \quad \text{and} \quad T_2 = (t_1, r^{-1}, h_1)$$

where r^{-1} is the inverse relation of r. These two triples are considered redundant because they represent the same semantic relationship in different directions. This redundancy can be eliminated as follows:

$$\text{BR}(T_1, T_2) = \text{True if } T_2 = (t_1, r^{-1}, h_1) \tag{3}$$

Transitive Redundancy (TR): For two triples:

$$T_1 = (h_1, r_1, t_1) \quad \text{and} \quad T_2 = (t_1, r_2, t_2)$$

we can infer the third triple:

$$T_3 = (h_1, r_1 \circ r_2, t_2)$$

where $\circ$ represents the composition of relations. If T_3 exists in the graph, then T_2 is considered redundant. The redundancy can be formalized as:

$$\text{TR}(T_1, T_2, T_3) = \text{True if } T_3 \in \text{KG} \tag{4}$$

Functional Dependency Redundancy (FDR): If each entity corresponds to a unique relation $r(e)$, i.e., for each entity e, there exists a unique tail entity t such that:

$$r(e) = t$$

and if there are multiple triples with the same head entity h_1, the same relation r, and different tail entities t_1 and t_2, then redundancy exists and should be removed. The redundancy is formally expressed as:

$$\text{FDR}(h_1, r, t_1, t_2) = \text{True if } t_1 \neq t_2 \tag{5}$$

Entity Complexity: Entity complexity is calculated based on two factors: the **single-document frequency** and the **specialization ratio**.

Single-Document Frequency: The information content of an entity e is inversely related to its frequency of occurrence in the document. This is calculated using the following formula, where f_e is the frequency of the entity e in the document, and $P(e)$ is its probability of occurrence:

$$I_e = -\log P(e) = -\log\left(\frac{f_e}{\text{total tokens}}\right) \tag{6}$$

Here, total tokens represents the total number of tokens in the document, and $P(e)$ is the probability of entity e occurring in the document.

To normalize this value, we apply a bounded range:

$$I_e^{\text{norm}} = \min\left(\max\left(I_e, 0\right), \text{bound}_{\text{max}}\right) \quad (7)$$

where $\text{bound}_{\text{max}}$ is a user-defined upper limit to ensure the information content stays within a reasonable range.

Specialization Ratio: The specialization of an entity e is determined by the ratio of its frequency in a specialized corpus (f_{dom}) to its frequency in a general corpus (f_{gen}). This ratio reflects how specialized the entity is within its domain. The specialization ratio is given by:

$$w_e = \frac{f_{\text{dom}}}{f_{\text{gen}}} \quad (8)$$

To normalize this value, we take the logarithm of the ratio and apply bounding:

$$w_e^{\text{norm}} = \min\left(\max\left(\log_2 w_e, 0\right), \text{bound}_{\text{max}}\right) \quad (9)$$

where $\text{bound}_{\text{max}}$ is again a user-defined upper bound for the log-transformed specialization ratio.

Entity Complexity: The final entity complexity is the sum of the normalized single-document frequency and specialization ratio. Both components are weighted such that their sum equals 1. The complexity of entity e is computed as:

$$C_e = w_1 \cdot I_e^{\text{norm}} + w_2 \cdot w_e^{\text{norm}} \quad (10)$$

where w_1 and w_2 are the weights assigned to the single-document frequency and specialization ratio, respectively, and $w_1 + w_2 = 1$.

The resulting complexity value C_e is in the range [0, 1], with higher values indicating more complex entities that are either more specialized or less frequent within the document.

C Theoretical Analysis

C.1 Online Construction Complexity

The online construction phase consists of triple generation, multi-type redundancy removal, and definition expansion. Let the corpus size be N, representing the total number of processed textual units.

Triple Generation. The text is segmented and processed sequentially by an LLM, and each chunk yields a bounded number of factual triples. Thus, the total time complexity is $O(N)$.

Semantic Redundancy Removal (R_s). Each entity or relation requires embedding computation with cost $O(N)$, followed by FAISS-based similarity clustering. Since FAISS employs approximate nearest neighbor search, the average clustering complexity is nearly linear, $O(N \log N)$.

Structural Redundancy Removal (R_f, R_b, R_t).

- **Functional Redundancy (R_f)**: Building mappings $(h, r) \to \{t\}$ takes $O(N)$.
- **Bidirectional Redundancy (R_b)**: Sorting or hashing entity pairs for deduplication costs $O(N \log N)$.
- **Transitive Redundancy (R_t)**: Each triple is checked against adjacent nodes, leading to an average complexity of $O(N \log N)$ in sparse graphs.

Definition Expansion. Because both the expansion depth and the per-entity expansion count are bounded, and only entities with complexity above the threshold are expanded, the worst-case complexity remains $O(N)$, with significantly lower cost in typical cases.

Overall Analysis. The total expected complexity of online construction is dominated by the semantic clustering stage, resulting in an overall cost of approximately $O(N \log N)$. This ensures that the construction process remains computationally feasible even for large-scale corpora.

C.2 Online Inference Complexity Analysis

We adopt a SPARQL-style query execution model to analyze the inference complexity in the KG-RAG framework. Each query corresponds to retrieving a subgraph that best matches the semantic structure of the query pattern. The overall retrieval efficiency primarily depends on the number of candidate nodes and edges participating in query matching and join operations.

After processing by the Redundancy Removing Component, both the number of entities ($|V|$) and relations ($|E|$) are reduced. Semantic redundancy removal merges semantically equivalent entities, resulting in a smaller $|V'|$, while structural redundancy removal eliminates redundant or derivable

edges, yielding $|E'| < |E|$. The total cost of subgraph matching is approximately proportional to the number of nodes and edges that must be searched:

$$T_{\text{retrieval}} \propto |V'| + |E'|. \tag{11}$$

Because both $|V'|$ and $|E'|$ are smaller than their original values, the retrieval time decreases correspondingly. From the perspective of SPARQL query processing, the reduction of redundant entities narrows the variable binding space, while the elimination of unnecessary edges reduces the cost of join operations. Therefore, the Redundancy Removing Component effectively improves inference efficiency while *striving to preserve factual completeness*.

In contrast, the Definition Enrich Component introduces additional definitional information through an indexed structure. Each entity is associated with its enriched definitions and extended entities via pre-built indices, allowing constant-time lookup during reasoning. Since the definitions form a tree-like hierarchy with bounded depth, the corresponding retrieval cost remains nearly negligible. This additional structure enriches semantic interpretation without notably affecting inference latency.

In summary, the Redundancy Removing Component strives to preserve factual completeness while reducing storage overhead and improving retrieval efficiency. The Definition Enrich Component slightly increases retrieval time but supplements entities with hierarchical definitions, thus enhancing the factual precision and interpretability of KG-QA. In practical scenarios, users can flexibly adjust the thresholds of both components to balance efficiency, storage, and reasoning accuracy according to their specific requirements.

D SpaCy

D.1 Knowledge Graph Construction

As a baseline, we adopt a rule-based triple extraction pipeline built on dependency parsing. This pipeline mainly relies on traditional NLP tools such as spaCy (Honnibal et al., 2020) for tokenization, part-of-speech tagging, dependency parsing, and noun phrase chunking. Given an input document, the text is first segmented into sentences and processed using a dependency parser. For each sentence, predicate candidates are identified based on dependency labels and part-of-speech tags, including main verbs and certain clausal verbs (e.g., ROOT, conj, xcomp, ccomp). Weak or semantically underspecified verbs (e.g., be, have, do) are excluded to reduce low-information relations. Subject and object entities are determined through dependency relations such as nsubj, nsubjpass, dobj, attr, and prepositional objects (prep–pobj). To improve entity span quality, entities are expanded using noun phrase chunks or dependency subtrees, with an upper bound on entity length to avoid overly long spans. Conjunctive structures are handled by collecting coordinated subjects or objects within the same syntactic scope.In addition to verb-mediated relations, copular constructions are explicitly handled. For sentences where the root is a noun or adjective with a copular verb, subject–predicate relations are extracted in the form of normalized copular relations, while a predefined set of weak predicates is filtered out. To suppress noise from generic copular statements, copular triples are further filtered by a minimum frequency threshold.After initial extraction, several post-processing steps are applied. First, the number of tail entities per (head, relation) pair is capped to limit overly prolific relations. Second, copular triples not meeting the frequency threshold are removed. Finally, near-duplicate triples are eliminated using a normalization-based matching strategy that compares lemmatized entity forms and relation labels. Through this process, the pipeline extracts 7,264 candidate triples from the raw text and retains a final set of 5,679 triples after filtering and deduplication.

In contrast, our approach incorporates large language models (LLMs) to perform coreference resolution and triple extraction, producing a total of 9,350 triples. Manual inspection shows that the entities and relations extracted by both methods are largely grounded in the original text and achieve comparable accuracy. Due to the absence of manually annotated ground truth, exact recall cannot be computed. We therefore compare the two approaches from a structural perspective of the resulting knowledge graphs. The results indicate that the knowledge graph constructed by the LLM-based method contains more nodes and exhibits higher graph density, enabling broader coverage of entities and relations.

This observation is highly consistent with the conclusions reported in (Wadhwa et al., 2023), which show that, for relation extraction and knowledge graph construction, LLM-based methods outperform traditional rule-based or tool-driven ap-

proaches (e.g., spaCy) by improving the overall effectiveness of the constructed graphs while maintaining comparable accuracy.

D.2 Knowledge Graph Redundancy Removal

We adopt a rule-based deduplication strategy with lemmatization to normalize and merge extracted triples. Specifically, each entity mention is first normalized by lowercasing, lemmatization, and punctuation removal, while preserving the original token order (Manning et al., 2014) . Deduplication is then performed by comparing triples that share the same relation: two triples are considered redundant if both their head and tail entities are identical after normalization, or if one normalized entity string is a substring of the other. This design allows the baseline to handle simple morphological variations (e.g., singular/plural forms) as well as basic cases of entity inclusion (Lin et al., 2020) . When redundancy is detected, the method retains the more general triple by favoring the entity mention with fewer tokens. Under this baseline approach, the graph size is reduced from 5,832 to 5,448 triples, corresponding to a reduction rate of approximately 6.6%. These results suggest that rule-based methods grounded in surface-form normalization are effective for removing explicit and lexically similar redundancies, but remain limited in their ability to capture deeper semantic equivalence beyond string-level similarity.

In contrast to traditional rule-based approaches, our method introduces a large language model (LLM) for deduplication, achieving a graph size reduction of over 20%. Compared with the baseline, the LLM is able to better capture deep semantic consistency between entities and relations, enabling the detection of semantic redundancies that are difficult to handle with handcrafted rules. Experimental results demonstrate that the LLM-based deduplication strategy significantly improves compression efficiency while preserving the semantic integrity of the graph, and clearly outperforms the rule-based baseline.

E Knowledge Graph RAG Process

Algorithm 1 Hierarchical Knowledge Subgraph Retrieval

Input: Natural language query $Q = \{q_1, q_2, \ldots, q_n\}$, Knowledge graph $G = (E_G, R_G)$, Embedding model ϕ, Depth T, Top-K threshold
Output: Retrieved subgraph $S_T = (V_S, E_S)$

$\epsilon_Q, R_Q \leftarrow \text{LLM\_EXTRACT}(Q)$ ▷ Extract entities and relations
$P \leftarrow \text{BUILD\_PATH}(\epsilon_Q, R_Q)$ ▷ Construct initial path $e_1 \to r_1 \to e_2 \to \ldots$
$v_h \leftarrow \phi(e_1)$ ▷ Embed head entity
$e_{\text{align}} \leftarrow \arg\max_{e_i \in E_G} \left(\frac{v_h \cdot \phi(e_i)}{\|v_h\| \|\phi(e_i)\|} \right)$ ▷ Max cosine similarity
$S_0 \leftarrow \{e_{\text{align}}\},\ B_0 \leftarrow \{e_{\text{align}}\},\ t \leftarrow 1$
while $t \leq T$ **and** $|S_{t-1}| <$ size_threshold **do**
  $N_{\text{all}} \leftarrow \emptyset$
  for each $b \in B_{t-1}$ **do**
    $N(b) \leftarrow \text{GETNEIGHBORS}(b, G)$ ▷ Fetch 1-hop neighbors
    $N_{\text{filtered}}(b) \leftarrow \text{LLM\_RELEVANCE}(N(b), Q)$ ▷ Semantic pruning
    $N_{\text{topK}}(b) \leftarrow \text{TOPK}(N_{\text{filtered}}(b), K)$ ▷ Retain top-K
    $N_{\text{all}} \leftarrow N_{\text{all}} \cup N_{\text{topK}}(b)$
  end for
  $S_t \leftarrow S_{t-1} \cup N_{\text{all}}$
  $B_t \leftarrow (B_{t-1} \setminus \{b\}) \cup N_{\text{all}}$ ▷ Update boundary
  $t \leftarrow t + 1$
end while
return S_T ▷ Pruned subgraph

Process Explanation

This algorithm implements Knowledge Graph Retrieval-Augmented Generation (KG-RAG) with four key phases:

1. **Input Parsing:** An LLM parses natural language query Q to extract candidate entities (ϵ_Q) and relation chains (R_Q), forming an initial reasoning path.

2. **Entity Alignment:** The head entity (e_1) is embedded into a vector space and aligned to the KG entity with maximum cosine similarity, resolving lexical variations.

3. **Hierarchical Expansion:** Starting from the aligned entity, the subgraph expands iteratively:

   - At each depth t, retrieve neighbors of boundary nodes
   - LLM evaluates neighbor relevance to Q for semantic filtering
   - Top-K relevant neighbors are added to

the subgraph

Expansion terminates at depth T or upon reaching size threshold.

4. **Output:** The pruned subgraph S_T contains KG entities and relations contextual to Q, serving as structured knowledge for downstream reasoning.

The hierarchical expansion with semantic pruning balances efficiency (complexity $O(T \cdot K \cdot |B|)$) and relevance, outperforming naive neighborhood sampling.

F "is_A" Baseline for Definition Enrichment

In this appendix, we describe the "is_A" baseline method used for entity definition enrichment, which leverages Retrieval-Augmented Generation (RAG) to generate entity definitions. This method utilizes the knowledge available in external resources, such as Wikidata, to obtain related entities and their definitions, which are then used to enrich the target entity's definition.

The core idea of the "is_A" baseline is to retrieve similar entities from an external knowledge source (e.g., Wikidata) and use their definitions to generate a definition for the target entity. This process involves:

- **Entity Definition Retrieval**: For each entity in the knowledge graph, the RAG-based component queries a knowledge source, such as Wikidata, to retrieve entities that are semantically similar to the target entity.
- **Definition Extraction**: The retrieved similar entities' definitions are collected from the external source.
- **Definition Generation**: The LLM uses the definitions of related entities to generate a corresponding definition for the target entity. This is typically done by identifying the most relevant aspects of the related entities and combining them into a concise definition.
- **Integration**: The generated definition is then integrated into the knowledge graph as the entity's enriched definition.

The "is_A" method leverages existing definitions of related entities and is a straightforward approach to providing context for entities within a knowledge graph. However, while it can be effective for entities with well-established and clear relationships, it may not provide sufficient detail for entities that require more nuanced or domain-specific definitions.

In contrast, the proposed recursive definition enrichment approach in this paper enhances the entity definitions by hierarchically expanding the definitions based on the entity's complexity, which allows for more contextually accurate and precise definitions. This improvement over the "is_A" method is particularly beneficial for handling complex or domain-specific entities where simple retrieval-based methods may fail to capture the full meaning.

G QA Pair Generation Templates

To facilitate consistent and scalable evaluation, we design a fully automated framework that constructs QA pairs directly from the knowledge graph. Three distinct QA categories are defined: (i) structure-only QA pairs that rely solely on in-graph structural relations, (ii) definition-only QA pairs that focus on definitional triples, and (iii) Complex QA pairs that combine structural and definitional information.

G.1 Structure-only QA pairs.

For the structure-only QA pairs, the framework first filters the raw triples. For each triple, a large language model (LLM) infers the tail entity from the given head and relation. The triple is retained only if the embedding similarity between the generated and original tails exceeds 0.5, ensuring that the resulting reasoning chains are well-posed and lead to unique, valid answers. From the filtered graph, we construct multi-hop chains of two to three intermediate relations as the basis for QA generation. The head entity and intermediate relations are presented to the LLM, which generates a question asking what entity is reached after sequentially applying these relations. The correct answer is the chain's tail entity.

An example of a two-hop path [E1, R1, E2, R2, E3] produces a question involving E1, R1, and R2, and the expected answer is E3.

Prompt Template of Structure-only QA Pairs

Produce exactly one natural, concise question wrapped inside <question>...</question>.
Only output the question inside the tags—no explanations, no labels.

Use a multi-hop logic with neutral referents:
Hop 1 (prerequisite): from '{E1}' via relation '{R1}', there is an intermediate (use 'one thing', 'something', 'that result', or 'the intermediate outcome').
Hop 2..(k-1): from that intermediate via relation '{R2..R{k-1}}', obtain the next outcome.
Final Hop: from the current intermediate via relation '{Rk}', ask what is ultimately obtained (this is the question).

Keep the exact order of relations.
Do NOT reveal any intermediate or final entity names. The answer should be a single word or a short phrase.

Style references (do not copy verbatim; write a natural sentence of your own):
<question>'{E1}' first applies {R1} to obtain one thing; if that result then undergoes {R2}, what is obtained in the end?</question>
<question>Assume '{E1}' yields something via {R1}. When that intermediate outcome subsequently experiences {R2}, what does it produce?</question>
<question>Starting from '{E1}', apply {R1; then R2; ...; then Rk} to the intermediate each time; what is ultimately obtained?</question>

Now write your single question and the answer:

Figure 5: Prompt template of Structure-only QA pairs.

G.2 Definition-only QA pairs.

For definition-based QA pairs, we first enhance the KG using the RAG-based Definition Enrich Component, generating definition triples in the form [entity, relation, definition]. These triples are concatenated into reasoning chains similar to the structure-only case, except that auxiliary relations such as copulas are removed to maintain semantic clarity. The LLM is then prompted to produce natural questions where the answer depends on the definitional relation between entities.

Prompt Template of Definition-only QA Pairs

Produce exactly one natural, concise question wrapped inside <question>...</question>.
Only output the question inside the tags—no explanations, no labels.

Requirements:
Express the hop sequence explicitly as: {E1} {R1} something; That something {R2} something; The intermediate result {R3} something; ...
After listing the hops, directly ask what the final 'something' is.

Keep the exact order of relations and ensure the sentence is fluent and grammatical.
Do NOT reveal any intermediate or final entity names; the expected answer is a noun or noun phrase that defines or categorizes '{E1}'.

Examples (do not copy verbatim):
<question>{E1} {R1} something; That something {R2} something; The intermediate result {R3} something; ...; what is that final 'something'?</question>
<question>{E1} {R1} something; That something {R2} something; The intermediate result {R3} something; .... In the end, what is the last 'something'?</question>
<question>{E1} {R1} something; That something {R2} something; The intermediate result {R3} something; ...; ultimately, what is the final 'something' obtained?</question>

Now write your single question and the answer:

Figure 6: Prompt template of Definition-only QA pairs.

G.3 Complex QA pairs.

Complex QA pairs integrate both structure-only and definition-only QA types. Specifically, two or three structural triples are combined with one or two definitional triples, forming a mixed chain. Conceptually, a reasoning path begins with a concrete entity, proceeds through multiple relational hops to reach a complex or abstract concept, and then follows a definitional edge to generate an interpretable description. The QA pair thus assesses both reasoning and interpretive understanding.

For the three categories of QA prompt templates, the input consists of preprocessed triple chains, and the output comprises the questions and answers generated by the large language model.

Table 4: Comparison of Triple Count and Generation Time Across Different Models and Chunk Sizes

Model	chunk size	1000	5000	10000	50000
GPT-4o-mini	Triples Count	9350	6264	4797	3002
	Generation Time (min)	110	90	74	63
GPT-4.1	Triples Count	7479	6023	3787	2336
	Generation Time (min)	94	80	66	42
GPT-5	Triples Count	-	2449	1158	274
	Generation Time (min)	-	423	297	145
GPT-O3-mini	Triples Count	5547	2118	1079	187
	Generation Time (min)	599	394	277	130

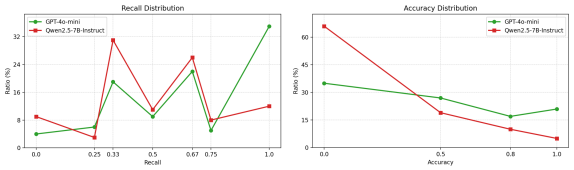

Figure 8: Distribution of recall and accuracy across 100 QA pairs generated by *GPT-4o-mini* and *Qwen-2.5-7B*.

H Further Experimental Analysis

Prompt Template of Hybrid QA Pairs

Intra-graph part:
Write exactly one natural, concise sentence wrapped inside <question>...</question> that describes a multi-hop path without asking any question.
Output only the sentence inside the tags, no extra text.

Use neutral referents like 'something', 'that result', or 'the intermediate outcome' for all intermediate and final entities.
The sentence should describe this path: Starting from '{E1}', via relation '{R1}', there is something. / from '{E1}' via '{R1}' there is something; that result then undergoes '{R2}' to yield something; the intermediate outcome then undergoes '{R3}' to yield something; ... .

Do not ask who/what the final entity is; just describe up to obtaining 'something' at the end.

Definition part:
Examples (do not copy verbatim):
<question>This something {R_ext1} what?</question>
<question>This thing {R_ext1} something, which {R_ext2} something, ..., which {R_extk} something; what is that final 'something'?</question>

Now write your single question and the answer:

Figure 7: Prompt template of Complex QA pairs.

H.1 Data Distribution Analysis

As shown in Figure 8, we present the distribution of recall and accuracy for the 100 QA pairs generated under the main experimental configuration, using the *GPT-4o-mini* and *Qwen-2.5-7B* language models.

From the figure, we observe that *GPT-4o-mini* exhibits a recall distribution concentrated around 0.67 and 1.0, while its accuracy distribution is relatively smooth. This suggests that the subgraphs

Table 5: Evaluation results of recall and accuracy across different QA types, KG configurations, and LLM backbones.

	GPT-4.1						Qwen2.5-7B-Instruct					
	Recall			Accuracy			Recall			Accuracy		
Configuration	Struct.	Def.	Complex	Struct.	Def.	Complex	Struct.	Def.	Complex	Struct.	Def.	Complex
Full KG	0.33	0.24	0.44	0.21	0.16	0.07	0.34	0.29	0.57	0.05	0.13	0.10
Red.-KG + Rec. Def. (Ours)	0.44	0.50	0.53	0.52	0.44	0.53	0.31	0.35	0.60	0.32	0.29	0.52

retrieved and the answers inferred from them are of relatively high quality. In contrast, *Qwen-2.5-7B* has a recall distribution concentrated around 0.33 and 0.67, with its accuracy distribution showing a continuous decline. More than 60% of the QA pairs generated by *Qwen-2.5-7B* have an accuracy of 0, indicating that while its retrieval performance is decent, its ability to reason from the retrieved subgraphs is relatively weaker.

H.2 Analysis of Model Selection

In this subsection, we compare the performance of fast models and thinking models in knowledge graph construction. For fast models, we selected *GPT-4o-mini* and *GPT-4.1*, while for thinking models, we chose *GPT-5* and *o3-mini*. As shown in Table 4, we recorded the runtime and the number of triples generated by these models at different chunk sizes.

The results show that *GPT-4o-mini* and *GPT-4.1* are able to respond quickly and generate a large number of triples. In contrast, *O3-mini* and *GPT-5* have longer response times and generate fewer triples.

This demonstrates that thinking models tend to select only the most core triples, ignoring less critical facts. Moreover, thinking models require significantly more time and resources, making them less suitable for large-scale triple extraction tasks.

H.3 Evaluation with GPT-4.1

In this subsection, we repeat the evaluation using *GPT-4.1* for both the knowledge graph (KG) and question-answer (QA) pair generation, following the same methodology as in the main experiment with *GPT-4o-mini*.

As shown in Table 5, compared to the main experiment, the improvements for Complex QA pairs are more pronounced in the *GPT-4.1* based evaluation. This is due to *GPT-4.1*'s stronger reasoning capabilities compared to *GPT-4o-mini*, allowing it to handle more complex reasoning tasks more effectively.

This suggests that for experiments requiring higher reasoning abilities, *GPT-4.1* is a better choice.

I Potential Risks of Our Work

While our work focuses on improving knowledge graph construction and LLM-based QA tasks, we acknowledge that potential risks could arise from the misuse of automated systems, such as generating biased or incorrect information if the underlying models are trained on skewed data. Additionally, there is the challenge of ensuring the generalizability and robustness of our approach across diverse domains. These concerns will be addressed in future work through further testing and refinement.