# FrOG: Framework of Open GraphRAG

Jaycent G. Ongris[1,†], Eduardus Tjitrahardja[1,†], Fariz Darari[1] and Fajar J. Ekaputra[2]

[1]Faculty of Computer Science, Universitas Indonesia, Depok 16424, Indonesia
[2]Institute of Data, Process, and Knowledge Management, WU, Vienna 1020, Austria

**Abstract**

The rise of large language models (LLMs) has advanced information retrieval, yet issues like limited knowledge updating, lack of transparency and interpretability, as well as hallucinations persist. Retrieval-augmented generation (RAG) addresses these problems, though it still lacks interpretability due to reliance on opaque vector-based representations. Our work presents a RAG framework using a knowledge graph (KG) as the primary knowledge base to address this problem, relying solely on open-source components to enable user customization. Our pipeline comprises multiple stages: (*i*) a translation module for multilingual support, (*ii*) entity linking, (*iii*) knowledge retrieval through verbalized triples or SPARQL query generation, and (*iv*) answer generation, which incorporates ontology (properties and classes) retrieval. We evaluate our system on Wikidata, DBpedia, and a domain-specific KG. With the optimal configuration determined through an ablation study, the system achieves Jaccard similarity scores of 0.458, 0.517, and 0.976 for each respective KG. The ablation study further reveals that ontology retrieval is the most crucial component in providing context to the LLM in generating SPARQL queries.

**Keywords**

Large Language Models, Knowledge Graphs, Retrieval-Augmented Generation, GraphRAG

## 1. Introduction

Recent advancements in large language models (LLMs) have transformed information retrieval (IR) [1], powering services like ChatGPT, Gemini, and Copilot for tasks such as question-answering (QA) and text summarization. LLMs excel in IR due to their ability to semantically understand and generate natural language by leveraging knowledge internalized in their parameters [2]. While fine-tuning and in-context learning can further enhance their adaptability, LLMs face challenges like knowledge update difficulties, lack of transparency and interpretability, as well as *hallucination* [3], raising concerns about their reliability in critical domains.

To overcome these problems while still maintaining the excellent performance of LLMs, *retrieval-augmented generation* (RAG) was introduced [3]. RAG incorporates external non-parametric memory in the form of a retriever module that fetches relevant information. The retriever module allows expansion or update of knowledge as the knowledge is not embedded in the parameters. Moreover, this module also allows direct inspection of the retrieved data, partially addressing the black-box problem. While RAG does not guarantee that hallucinations will disappear entirely, it reduces them by not depending solely on the knowledge embedded within the LLMs.

The original naïve RAG approach, as proposed in [3], employs Dense Passage Retriever (DPR) as its non-parametric retrieval module [4]. This method encodes both queries and documents into dense embeddings, such as those generated using $\text{BERT}_{\text{BASE}}$, as implemented in [3]. The maximum inner product search (MIPS) algorithm is then used to retrieve the top-$K$ most relevant documents for a given query. However, despite outperforming traditional scoring algorithms like BM25 [4], DPR still suffers from the black-box problem due to the inherent opacity of dense embeddings, which are generally less interpretable. Additionally, splitting large documents into smaller segments for encoding (i.e.,

*4th International Workshop on LLM-Integrated Knowledge Graph Generation from Text (Text2KG) Co-located with the Extended Semantic Web Conference (ESWC 2025)*

[†]These authors contributed equally.

jaycent.gunawan@ui.ac.id (J. G. Ongris); eduardus.tjitrahardja@ui.ac.id (E. Tjitrahardja); fariz@ui.ac.id (F. Darari); fajar.ekaputra@wu.ac.at (F. J. Ekaputra)

chunking) can lead to the loss of contextual information and disrupt the coherence of the original content, especially when related information is spread across different chunks [5].

In order to address these issues, an alternative approach is to use a different knowledge base for retrieval instead of relying on a corpus of text chunks. One of the most versatile options is the knowledge graph (KG), which organizes information in a structured graph format. Leveraging a KG as the knowledge base offers several advantages: its entities are contextualized by their connections; its graph structure supports graph theory algorithms (e.g., community detection) for extracting insights; and it enables reasoning over retrieval results through graph walks that trace the connections between entities [6].

Given the interpretability and transparency challenges faced by most LLMs and naïve RAG, we introduce **FrOG: Framework of Open GraphRAG**[1]. FrOG is a RAG system that utilizes a KG as the primary knowledge base and relies entirely on open-source components. This approach enables direct tracing of retrieved knowledge, enhancing the system's reasoning process with greater transparency and interpretability. Moreover, we leverage off-the-shelf open-source LLMs, allowing users to implement the system directly without fine-tuning, which would otherwise require substantial computational resources. By using open-source components, we enhance accessibility and flexibility, making it easier to customize and improve the system for domain-specific applications. To narrow our focus, we limit our research to factoid questions and define our research questions (RQs) as follows.

**RQ1** How can RAG be implemented using a KG as the primary knowledge base (that is, GraphRAG) while relying only on open-source components?

**RQ2** Which open-source LLM serves as the best foundation for the GraphRAG architecture and achieves the highest answer accuracy?

**RQ3** Which specific component within the architecture plays the most critical role in ensuring accurate answer generation?

Our main contribution is the design and development of a GraphRAG pipeline built entirely with open-source components. We evaluate our pipeline across multiple KGs, including Wikidata, DBpedia, and a domain-specific KG, using various open-source LLMs to identify the best-performing model. Our results indicate that Qwen2.5 7B (instruction-fine-tuned) achieves the highest accuracy in the optimal configuration, with Jaccard similarity scores of 0.458, 0.517, and 0.976 for Wikidata, DBpedia, and the domain-specific KG, respectively. A demonstration of the system's capabilities in answering various questions from the respective KG can be viewed here[2]. Additionally, our findings highlight the retrieval of relevant KG resources—specifically entities, properties, and classes—as the most crucial component, as it provides essential context to the LLM and significantly enhances answer quality.

The rest of the paper is organized as follows: Section 2 reviews related work, Section 3 presents the system architecture, Section 4 details the experimental setup, Section 5 discusses the evaluation results, and Section 6 provides the conclusions and future work.

## 2. Related Work

GraphRAG is a retrieval-augmented generation (RAG) system that leverages KG as the knowledge base. The implementation involves retrieving graph elements that contain knowledge pertinent to the given query from a graph database [7]. GraphRAG provides several advantages over naïve RAG, including the ability to capture connections between entities for a more comprehensive retrieval of relational information, represent knowledge in a structured and efficient format rather than lengthy passages, and understand a broader context, enabling effective *query-focused summarization* (QFS) tasks [7].

The key difference between naïve RAG and GraphRAG lies in their knowledge bases and retrievers. Generally, naïve RAG relies on textual data as the knowledge base and utilizes DPR as the retriever. In contrast, GraphRAG employs various types of retrievers to extract information from a KG. Specifically,

[1] https://github.com/Framework-of-Open-GraphRAG/FROG

[2] Demo: https://drive.google.com/drive/folders/1UPVF-2rXrrToSsWB84h8L18A_YrMn1g6?usp=drive_link

Peng et al. [7] classify graph retrievers into three main categories: *non-parametric retrievers*, *language model (LM)-based retrievers*, and *graph neural network (GNN)-based retrievers*. Non-parametric retrievers apply heuristic rules and traditional graph search algorithms, LM-based retrievers leverage LMs (e.g., RoBERTa [8]) for natural language understanding, and GNN-based retrievers use GNNs to effectively capture and interpret graph structures.

Several recent end-to-end RAG approaches have incorporated KGs into their pipelines. For instance, Microsoft GraphRAG [9] automatically constructs a KG from source documents by extracting entities, relationships, and claims using an LLM. It then performs hierarchical community detection to organize the graph into thematic subgraphs, with community-level summaries used during query time to generate globally coherent answers. Another example by Cao et al. [10], LEGO-GraphRAG, incorporates an extensive retrieval phase consisting of two modules: the optional *subgraph-extraction module* to narrow the graph search space and the essential *path-retrieval module* to identify reasoning paths connecting relevant entities to answer the query.

Beyond the aforementioned approaches which directly work on the KG level, another method is to perform retrieval by querying the graph directly, i.e., query-based GraphRAG, where the query language can be either SPARQL for RDF data model or Cypher for Neo4j. In this approach, query generation is the most critical task and can be performed manually, semi-automatically, using schema-based methods, or through deep learning (DL) [11]. Recent advancements in DL and LLMs have increasingly favored the latter. For instance, [11] introduced SGPT, which employs a stack of Transformer encoders to encode both linguistic features of a question and its subgraph information before feeding them into GPT-2 [12] to generate the appropriate query. Other methods [13, 14] require no training, relying solely on in-context learning. A similar approach to ours is SPINACH [15], except that it utilizes a proprietary LLM to generate SPARQL queries.

## 3. Pipeline Architecture

We aim to develop a pipeline for question-answering over KGs. Given a question as input, the pipeline retrieves relevant information from the KG and generates an answer as output. Our approach is built entirely on open-source components, including all tools and LLMs, ensuring transparency, reproducibility, and adaptability.

Our pipeline extends our prior work [16] by introducing new features, including multilingual support and the integration of an external vector database for retrieving relevant properties and classes needed for generating SPARQL queries. By incorporating the latter feature, we eliminate the need to explicitly include the entire set of resources in the prompt, which could otherwise degrade the LLM performance [17]. We also address the flexibility and scalability limitations identified in [16] by supporting different types of KGs, including both open and enterprise KGs. Last but not least, to further enhance system accuracy, particularly for simple questions (one-hop questions), we integrate text-based retrieval using verbalized triples. The architecture is illustrated in Figure 1, with each component's role (cf. blue boxes) explained below.

### 3.1. Translation

Since the target KGs are in English, we incorporate a translation component to convert user questions from their original language into English, ensuring compatibility. The translation process begins with language detection to check if the input query is already in English. If not, the query is automatically translated before advancing to the next stages. For implementation, we use the `googletrans`[3] library in Python, an open-source and lightweight text translation library that utilizes the Google Translate API.

[3] https://pypi.org/project/googletrans/

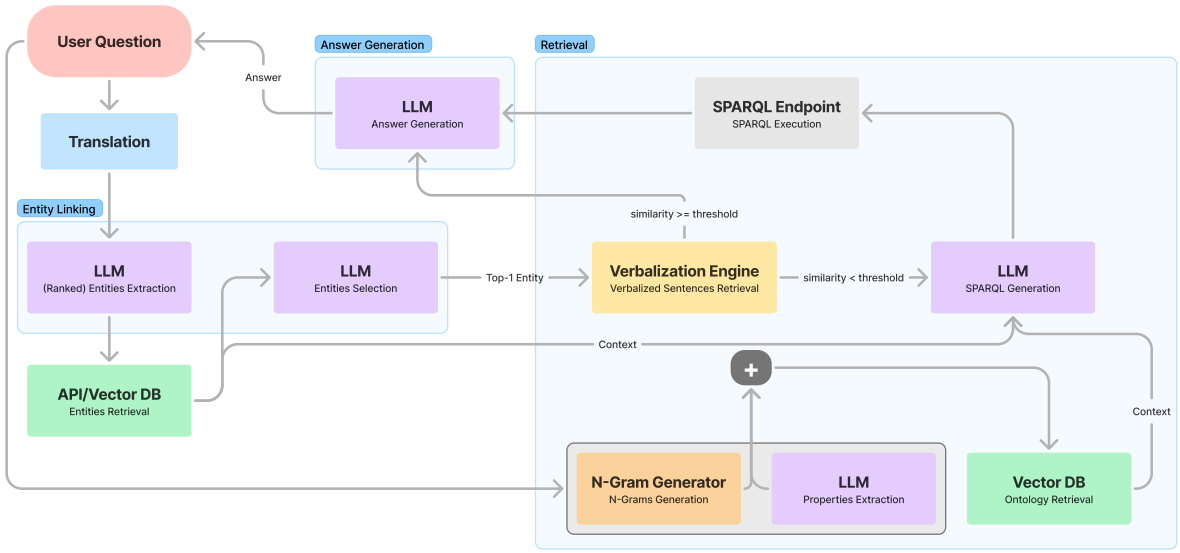


**Figure 1:** Our System's Pipeline Architecture

## 3.2. Entity Linking

This stage combines entity extraction, KG-specific entity retrieval, and entity disambiguation to map entities detected in the question to relevant KG-specific entities. The process begins by extracting entities from the question using an LLM with few-shot prompting. We instruct the LLM to rank the extracted entities by importance, prioritizing the most relevant one for the subsequent stage. Next, for each extracted entity, the top-$k$ KG-specific entities and their URIs are retrieved either via an API (e.g., Wikidata API[4] or DBpedia Lookup[5]) for open KGs or through semantic search (where entities are indexed in a vector database) for enterprise KGs. Finally, entity disambiguation is performed to identify the most relevant KG-specific entity needed to answer the question. If available, we also include the description of each KG-specific entity to provide more context to the LLM.

For example, consider the question "How many children does Lionel Messi have?" answered using the DBpedia KG with $k = 4$. The entity extraction stage is expected to detect `Lionel Messi` as the key entity. The top four KG-specific entities are then retrieved using DBpedia Lookup via keyword search, returning the following entities: `dbr:Lionel_Messi`, `dbr:Messi-Ronaldo_rivalry`, `dbr:List_of_international_goals_scored_by_Lionel_Messi`, and `dbr:List_of_career_achievements_by_Lionel_Messi`. Finally, the entity disambiguation stage should correctly identify `dbr:Lionel_Messi` as the most relevant KG-specific entity, as it represents "Lionel Messi", the football player who is expected to have children.

## 3.3. Retrieval

We distinguish between two types of questions, each requiring a different retrieval method. The first type, *simple questions*, can be answered using one-hop information, e.g., "When was Lionel Messi born?" and "What is the birthplace of Lionel Messi?" The second type, *complex questions*, may require multi-hop retrieval or aggregate queries, such as "How many children does Lionel Messi have?" and "When was the father of Lionel Messi's wife born?"

[4]https://www.wikidata.org/w/api.php
[5]https://lookup.dbpedia.org/api/search

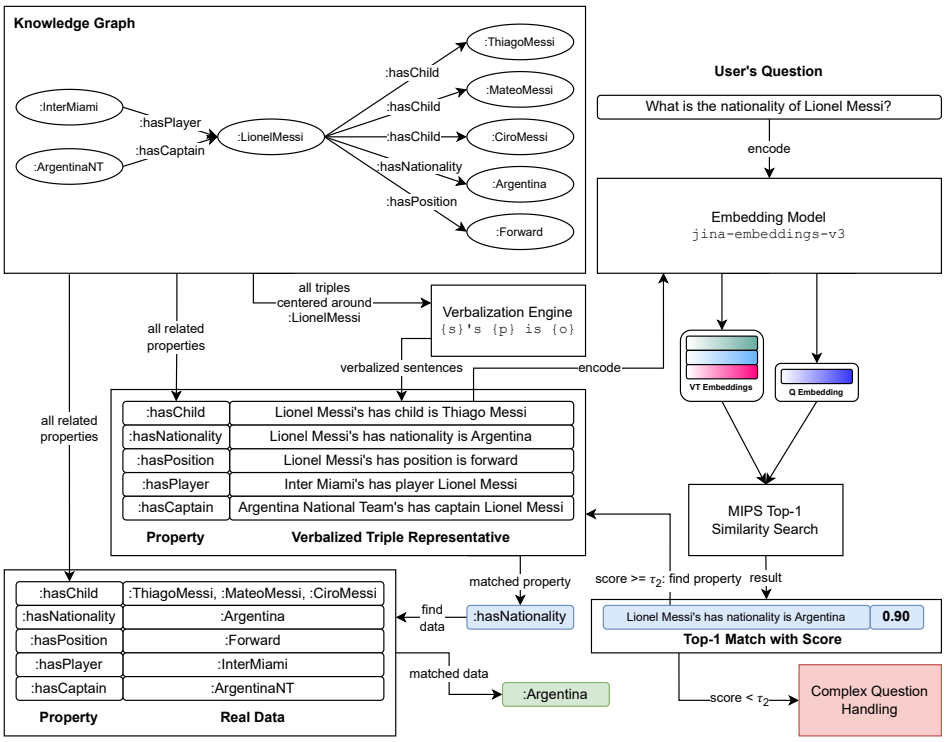


**Figure 2:** Simple Questions Answering Mechanism

### 3.3.1. Simple Questions

For simple questions, the system enhances efficiency by avoiding unnecessary SPARQL query generation and instead employs a text retrieval approach. This method is particularly effective for queries that can be answered using a single triple pattern from the KG.

The process begins by identifying the most relevant KG-specific entity from the detected entities in the question, as described in Section 3.2. Next, a single-hop breadth-first traversal is performed to retrieve the immediate neighboring entities of the identified KG-specific entity. The output consists of a set of triples representing the relationships between the entity and its neighbors. From this set of triples, *template-based verbalization* is applied to convert each triple `(s, p, o)` into a sentence using the rule: `{s}'s {p} is {o}`. To improve efficiency and reduce ambiguity, we ensure that each verbalized triple maintains a unique property when the given entity appears as both subject and object. If multiple triples share the same property but different objects, we generate a single representative sentence rather than multiple redundant ones. For instance, consider the KG in Figure 2, where the entity `:LionelMessi` has connections to three different entities through the property `:hasChild`. In this case, a single sentence is generated using the first matching triple, such as `Lionel Messi's has child is Thiago Messi`. Note that even if the generated sentence contains grammatical inaccuracies, the embeddings can still effectively capture the intended semantics.

The user's question and all verbalized triples are then encoded into embeddings to perform a semantic search with the QA objective, i.e., determining which generated sentence is the most probable answer to the user's question. The answer is returned only if the similarity score exceeds a predefined threshold; otherwise, the system proceeds with SPARQL query generation. A complete illustration of the verbalization-based retrieval mechanism is shown in Figure 2.

### 3.3.2. Complex Questions

For complex questions, we focus on generating accurate SPARQL queries using LLMs to retrieve answers. Since queries from complex questions often involve multiple entities, properties, and classes, the system

must first retrieve the relevant resources and explicitly include them in the prompt to provide context for generating SPARQL queries. Given that KG-specific entities have already been identified (cf. Section 3.2), the next step is to retrieve the most relevant properties and classes needed for query construction. To achieve this, we employ ontology retrieval, i.e., semantic search (implemented as Weaviate's hybrid search[6]) over sets of property and class labels available in the KG. However, directly comparing a long-form question like "What is the birth date of Tom Cruise?" to its corresponding short property label "birth date" is not effective. To address this, we split the question into smaller n-gram chunks. Additionally, we leverage an LLM to generate multiple property candidates based on the given question, as not all properties are explicitly mentioned. For instance, the system may fail to match the property "starred in" with any n-gram chunk from the question "What films does Tom Cruise play in?"

Once the relevant resources are retrieved, SPARQL query generation is performed using an LLM with few-shot prompting and chain-of-thought prompting to improve the model's reasoning capabilities. Finally, the generated SPARQL query is executed over the KG to obtain the necessary information needed for answering the question.

### 3.4. Answer Generation

The answer generation stage transforms the retrieved KG data into natural language responses. Since the retrieved data is often represented as URIs, the system first resolves these URIs into human-readable labels using SPARQL queries. Next, a generative LLM is employed to generate a corresponding natural language response, using the original user query and retrieved data serve as context. For multilingual support, the prompt explicitly includes the instruction `"Answer in {user's language} language"`. This ensures that responses are generated in the same language as the user's question.

## 4. Experimental Setup

In this section, we present details about the knowledge bases dan datasets used to evaluate the system, along with the evaluation metrics.

### 4.1. Knowledge Base

We evaluate our framework using two open KGs and one enterprise (local) KG. For the open KGs, we use Wikidata and DBpedia, while for the enterprise KG, we use the curriculum KG. In addition to the explanation we provide in this subsection, we also present the general comparison of each KG's characteristics in Table 1.

**Wikidata.** Wikidata [18] is an openly accessible, collaborative knowledge base developed by the Wikimedia Foundation. Wikidata stores information in a structured format similar to RDF triples [19], which can be retrieved via SPARQL queries. To utilize this knowledge base, we first obtain all properties (URIs and their human-readable labels) through SPARQL queries and store them in an offline vector database. These properties are later retrieved and used as context for SPARQL query generation (cf. Section 3.3.2). For other types of resources, i.e., entities and classes, we use the Wikidata API[4] to search for related resources.

**DBpedia.** DBpedia [20] is an open RDF KG constructed from Wikipedia[7]. Similar to Wikidata, DBpedia supports SPARQL querying for data retrieval. To extract properties and classes (URIs and their human-readable labels), we use DBpedia's T-Box Ontology[8]. However, a key limitation is that it only provides resources with the prefix `http://dbpedia.org/ontology/`, which also restricts our system's

[6] https://weaviate.io/blog/hybrid-search-explained
[7] https://www.wikipedia.org/
[8] https://www.dbpedia.org/resources/ontology/

| Characteristics | Wikidata | DBpedia | Curriculum KG |
|---|---|---|---|
| Triples Count | 1.67B[10] | 1.15B[10] | 874 |
| Category | Open KG | Open KG | Enterprise KG |
| Domain | General | General | Education |
| URI Representation | Cryptic | Human-readable | Human-readable |
| Host | Remote | Remote | Local (using `rdflib`) |

**Table 1**
Comparison of Each KG

coverage for this KG. Additionally, we utilize DBpedia Lookup[5] to search for relevant entities within the KG.

**Curriculum KG.** We use Curriculum KG [21] as a representative of a local/enterprise KG. This KG contains information about the Computer Science curriculum at Fasilkom UI (the Faculty of Computer Science, Universitas Indonesia). The KG is extracted from the publicly available curriculum guidebook[9] using GLiNER [22]. The original dataset is exported as a text file, with each row containing a triple representing information about a specific course, written in Indonesian. We apply additional preprocessing steps to translate the dataset into English and convert it into RDF triples. All resources (i.e., properties, classes, and entities) are stored in offline vector databases to facilitate semantic search.

## 4.2. Dataset

For open KGs (Wikidata and DBpedia), we utilize the QALD-9-Plus dataset [23], while for local KGs, we generate a custom KG-based QA dataset using our own dataset generator.

### 4.2.1. QALD-9-Plus

We evaluate our framework for Wikidata and DBpedia using QALD-9-Plus [23]. QALD-9-Plus is a multilingual knowledge graph question-answering (KGQA) benchmark dataset for Wikidata and DBpedia. It covers 10 languages (incl. English). The dataset consists of two splits: train split (408 unique questions) and test split (150 unique questions).

We use this dataset for two purposes: evaluation and benchmarking. For evaluation, we use the train split only with the following refinements. First, entries are filtered to exclude queries using resources outside the `dbo:` prefix or those returning only a single resource or numerical value. Next, each DBpedia entry is matched with an equivalent Wikidata entry to ensure consistency. In order to streamline evaluation, we downsample the dataset into 14 entries for in-context learning in SPARQL generation (cf. Section 3.3.2) and 65 entries for system evaluation.

To obtain a representative subset, we apply a cosine similarity downsampling technique, selecting the least similar entries to ensure diversity across question clusters. The process follows these steps:

1. Encode all sentences using a sentence embedding model (`all-mpnet-base-v2`[11]), which is based on the pre-trained MPNet [24] model.
2. Compute pairwise cosine similarities for all sentences.
3. Calculate the average cosine similarity for each sentence.
4. Sort sentences in ascending order based on their average cosine similarity scores.
5. Select the top-$k$ questions with the lowest similarity scores, ensuring that they are the least similar to the rest of the dataset.

[9] https://scele.cs.ui.ac.id/pluginfile.php/1279/block_html/content/buku_panduan_kur_2024_ilmu_komputer.pdf
[10] As of April 26th, 2025.
[11] https://huggingface.co/sentence-transformers/all-mpnet-base-v2

| Category | Triple Pattern |
|---|---|
| Simple 1 | `{ s p ?o . }` |
| Simple 2 | `{ ?s p o . }` |
| Complex 1 | `{ ?s p1 o1; p2 o2 . }` |
| Complex 2 | `{ ?s p1 ?o1 . ?o1 p2 o2 . }` |

**Table 2**
Supported Question Categories

For benchmarking against other graph-based RAG approaches, we use the full test split to compare system performance. However, due to DBpedia's constraints in our system (which only support queries within the `dbo:` prefix), we conduct benchmarking exclusively on the Wikidata version of the dataset to ensure comprehensive evaluation.

#### 4.2.2. KG-based Dataset Generator

Given the KG-agnostic nature of this framework, we provide a semi-automatic dataset generator that produces data for training and testing, particularly for local KGs. This system supports the generation of four types of questions, as described in Table 2, and also allows for the creation of queries with the `COUNT` clause, leveraging the same triple patterns. The output of this system is a dataset consisting of three columns: the question in natural language, the corresponding SPARQL query, and the category of the question. Below are the five main steps required to generate a question:

1. **Entity selection**. Randomly selects an entity (i.e., an instance of a class) from the given KG.
2. **Random walk**. Performs a random walk on the KG based on the question category, starting from the previously selected node. Users can optionally exclude specific schematic properties like `rdfs:domain` and `rdfs:range` to focus more on the concrete data graph.
3. **Resource label resolver**. Resolves natural language labels for properties and entities found in the triples.
4. **SPARQL query formation**. Wraps the retrieved triples into a SPARQL query using a pre-defined template based on the question category.
5. **Question generation**. Generates a natural language question from the generated SPARQL query and resource labels using an LLM. This process employs a zero-shot prompt template [25], leveraging off-the-shelf instruction-fine-tuned LLM.
6. **(Optional) Manual refinement**. Allows for manual review and adjustments of generated questions if they are deemed ambiguous.

### 4.3. Evaluation Metrics

We evaluate our system performance by by comparing the execution results of the generated SPARQL queries against the ground truth query execution results. The primary evaluation metric used is Jaccard Similarity, defined as in Equation 1 for two sets $A$ and $B$.

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

In our implementation, the original SPARQL query execution results preserve row ordering, particularly when queries include the `ORDER BY` clause. To ensure fair comparison, we transform both the predicted and ground truth execution results into sets before applying Jaccard Similarity. This transformation ensures that row ordering does not affect the evaluation. Additionally, each row is treated as a tuple, where elements are ordered alphabetically to maintain consistency. Thus, in Equation 1, $A$ and $B$ represent sets of tuples rather than ordered lists. An illustration of this evaluation schema is shown in Figure 3.

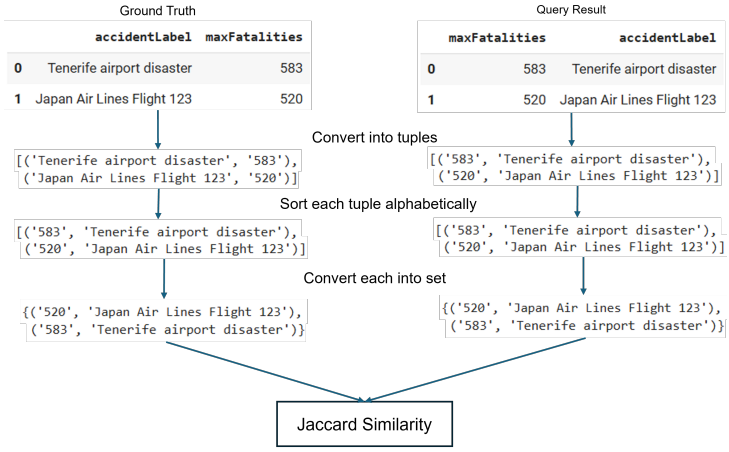


**Figure 3:** Evaluation Schema using Jaccard Similarity

Additionally, for benchmarking purposes, following [23], we also use F1 Macro as an evaluation metric. Using the same preprocessing steps as illustrated in Figure 3, we first compute precision and recall, as defined in Equation 2 and Equation 3, respectively, for a question $q$. The F1 score for $q$ is then calculated using Equation 4. Finally, for a set of questions $\mathbf{q} = \{q_1, q_2, ..., q_n\}$, F1 Macro is computed based on Equation 5.

$$\text{precision}(q) = \frac{\text{number of correct system answers for } q}{\text{number of system answers for } q} \tag{2}$$

$$\text{recall}(q) = \frac{\text{number of correct system answers for } q}{\text{number of gold standard answers for } q} \tag{3}$$

$$\text{F1}(q) = \frac{2 \times \text{precision}(q) \times \text{recall}(q)}{\text{precision}(q) + \text{recall}(q)} \tag{4}$$

$$\text{F1 Macro}(\mathbf{q}) = \frac{1}{n} \sum_{i=1}^{n} \text{F1}(q_i) \tag{5}$$

## 5. Results and Analysis

In this section, we present the experimental results obtained using the three aforementioned knowledge bases. We also provide benchmarking results comparing our system with several existing approaches and conduct an ablation study to identify the most crucial component within the pipeline. All LLMs referenced in this section correspond to their instruction-fine-tuned versions.

### 5.1. Experiment Results

Table 3 presents the evaluation results of four LLMs—Mistral NeMo [26] , LLaMA 3.1 8B [27], Qwen2.5 Coder 7B [28], and Qwen2.5 7B [29]—on datasets tailored to specific knowledge bases: the downsampled QALD-9-Plus for Wikidata and DBpedia, and the Curriculum KG for local KG experiments. All models were hosted locally. The experiments were conducted using the pipeline's default configuration, i.e., all features tested in the ablation analysis (cf. Section 5.3), such as verbalization, chain-of-thought prompting, and few-shot prompting were activated. Further error analysis related to the experiment is presented in our technical report [30].

| Configuration | Knowledge Base | Jaccard Similarity |
|---|---|---|
| Mistral NeMo 12B | Wikidata | 0.423 |
| LLaMA 3.1 8B | Wikidata | 0.427 |
| Qwen2.5 Coder 7B | Wikidata | 0.428 |
| **Qwen2.5 7B** | Wikidata | **0.458** |
| Mistral NeMo 12B | DBpedia | 0.450 |
| LLaMA 3.1 8B | DBpedia | 0.444 |
| Qwen2.5 Coder 7B | DBpedia | 0.442 |
| **Qwen2.5 7B** | DBpedia | **0.517** |
| **Mistral NeMo 12B** | Curriculum KG | **0.805** |
| LLaMA 3.1 8B | Curriculum KG | 0.778 |
| Qwen2.5 Coder 7B | Curriculum KG | 0.778 |
| **Qwen2.5 7B** | Curriculum KG | **0.805** |

**Table 3**
Results on the Downsampled QALD-9-Plus Dataset

**Wikidata.** In our evaluation, Qwen2.5 7B outperformed other models, achieving a Jaccard similarity of 0.458. It successfully handled entity-rich queries, such as "In which films directed by Garry Marshall was Julia Roberts starring?", where the model correctly mapped entities and properties, yielding an accurate SPARQL query. It also demonstrated strong performance on aggregation queries, such as "How many programming languages are there?", correctly forming a count-based SPARQL query. Furthermore, it effectively generated meaningful verbalizations, as seen in "In which programming language is GIMP written?", which allowed the system to correctly identify the relevant entity, `wd:Q15777` (C programming language).

We also observed several failure cases, which reveal the model's limitations. In "Give me all Australian metalcore bands," incorrect entity retrieval led to an incomplete query due to the model's lack of semantic understanding of implicit properties. Similarly, in "Show a list of soccer clubs that play in the Bundesliga," while relevant entities were retrieved, the model mistakenly used an incorrect property (`wdt:P6399`), resulting in an erroneous query. These findings emphasize the importance of precise entity-property retrieval and LLM prioritization, which is further analyzed in Subsection 5.3.4.

Moreover, we also tested the multilingual capabilities of Qwen2.5 7B on the Indonesian-translated Wikidata QALD-9-Plus, where its Jaccard similarity dropped to 0.362. This suggests that while the model can handle multilingual queries, its accuracy declined due to translation-induced paraphrasing. For example, the English question "Show me all the breweries in Australia." was translated into "Tunjukkan semua pabrik bir di Australia." and then back-translated as "Show all beer factories in Australia.", causing the system to fail to retrieve `wd:Q131734` ("brewery"). Despite these limitations, the results confirm the system's multilingual capability, particularly in Indonesian.

**DBpedia.** Qwen2.5 7B achieved the highest Jaccard similarity (0.517), outperforming other models on the DBpedia dataset. Its improved performance compared to Wikidata is likely due to DBpedia's human-readable URIs (e.g., `dbr:Australia` vs. `wd:Q408`), which facilitated entity retrieval. The model successfully generated accurate SPARQL queries, such as "Show a list of soccer clubs that play in the Bundesliga," and "How many programming languages are there?", correctly extracting entities, properties, and classes. Verbalization also performed well, as demonstrated in "In which programming language is GIMP written?", where the verbalized output, "GIMP's programming language is C (programming language)," achieved a similarity score of 0.74, accurately identifying `dbr:C_(programming_language)`.

Despite these improvements, challenges persisted, particularly in entity-property extraction. In "Give me all Australian metalcore bands," the model retrieved `dbr:Australia` and `dbr:Metalcore` but failed to extract `dbo:hometown` and `dbo:country` as it lacked semantic understanding of implicit relationships, leading to an incorrect query. Similarly, in "List all boardgames by GMT," the pipeline misinterpreted "GMT," returning entities like "Greenwich Mean Time" instead of `dbr:GMT_Games`, causing incorrect

query generation. These findings underscore DBpedia's advantages while also emphasizing the need for more precise entity-property alignment.

**Curriculum KG.** In this evaluation, Qwen2.5 7B and Mistral NeMo achieved the highest Jaccard similarity (0.805), with LLaMA 3.1 8B and Qwen2.5 Coder 7B following closely behind (0.778). The slight difference in performance was due to Qwen2.5 Coder 7B failing to answer "What courses have Research Methodology and Scientific Writing as prerequisites with a report as an evaluation method?" and LLaMA 3.1 8B struggling with "How many prerequisite courses does 'Algorithm Design and Analysis' have?", whereas Mistral NeMo and Qwen2.5 7B answered both correctly.

A closer analysis reveals that a recurring issue across all models was linked to the verbalization component. For example, in "How many evaluation methods of 'Internet of Things'?", the verbalized output "Internet of Things's has evaluation method is Task" had a similarity score of 0.636, exceeding the similarity threshold of 0.6 and incorrectly returning `http://example.org/group_project` and `http://example.org/task` instead of correctly aggregating them into the correct answer (2). This highlights a limitation in the verbalization process, where overly simplistic or mismatched interpretations lead to errors. Given these limitations, further evaluation is necessary, particularly in the verbalization ablation study (cf. Subsection 5.3.1), to better assess model performance and mitigate these challenges.

### 5.1.1. General Analysis

This subsection provides an analysis of the experimental results from the perspectives of both the LLMs and the knowledge bases.

**LLMs.** The experimental results indicate that Qwen2.5 7B consistently outperforms other models across datasets, achieving the highest Jaccard similarity scores on Wikidata (0.458), DBpedia (0.517), and the curriculum KG (0.805), where it performed on par with Mistral NeMo. This strong performance highlights Qwen2.5 7B's versatility in processing natural language instructions and generating diverse outputs, including free text, structured data, and code.

One likely reason for this performance is Qwen2.5 7B's extensive pre-training on a high-quality corpus comprising 18 trillion tokens, compared to only 15 trillion tokens used for LLaMA 3.1 8B. The Qwen2.5 corpus was curated through advanced data filtering and mixture strategies, balancing domain representation by downsampling content from overrepresented or low-quality domains and upsampling high-value sources such as scientific, technical, and academic texts. Additionally, the pre-training data was enriched with domain-specific code and mathematics datasets as used in the training for Qwen2.5 Coder and Qwen2.5 Math, alongside high-quality synthetic data generated and filtered using other large Qwen models. This diverse and expansive pre-training corpus has exposed Qwen2.5 7B to a broader range of linguistic patterns, factual knowledge, and contextual nuances, leading to improved accuracy. Furthermore, Qwen Team [29] reported that the instruction-fine-tuned 72B variant of Qwen2.5 significantly outperforms LLaMA 3.1 70B on multiple benchmarks, further reinforcing its superior ability to follow instructions and generate precise responses.

Interestingly, despite initial expectations, the Coder variant of Qwen2.5 7B underperforms compared to its general-purpose counterpart. Qwen2.5 Coder 7B was trained on a dataset where 70% of the data is code-related, covering 5.2 trillion tokens, while only 20% of its corpus consists of general text. While this specialization enhances its coding proficiency, it appears to hinder its broader understanding of natural language, which is crucial for SPARQL query generation. The generation process not only requires code synthesis but also a deep comprehension of user queries and their KG context, an area where the general-purpose Qwen2.5 7B model excels due to its more diverse training data.

For Mistral NeMo, drawing definitive conclusions is challenging due to the lack of detailed documentation on its training data. However, its relatively strong performance may be attributed to its larger parameter count (12B), compared to the other models (7B–8B). Larger models generally capture more complex patterns and relationships, which could contribute to improved performance. Without

further information on its dataset, this remains speculative, but model size likely plays a key role in its effectiveness.

**Knowledge Bases.** We identified two key factors affecting system performance from the perspective of knowledge bases:

1. **URI Representation:** The way entities are represented within a KG plays a crucial role in model performance. When URIs follow natural language conventions, as in DBpedia and the curriculum KG, models tend to perform better. Since LLMs are trained on natural language, human-readable URIs enhance entity linking and property retrieval. Conversely, cryptic or non-intuitive URIs make interpretation more difficult, reducing model accuracy.
2. **Size and Homogeneity:** The scale and complexity of a KG impact system performance. Larger and more diverse KGs (e.g., Wikidata and DBpedia) introduce a wider variety of entities, relationships, and question types. While this enhances versatility, it also increases query complexity, making SPARQL generation more challenging. In contrast, smaller and more homogeneous KGs like the curriculum KG limit the scope of possible queries, reducing query complexity and improving performance.

## 5.2. Comparison to Other Graph-based RAG Approaches

We evaluate the performance of our system on the QALD-9-Plus test set [23], comparing it against several established approaches. Specifically, we include SPINACH [15], QAnswer [31], Platypus [32], and DeepPavlov [33] in our comparison. Among these, we consider SPINACH as a representative of generative LLM-based systems, while the others are categorized as non-generative LLM-based systems.

Due to the limitation in handling resources outside the `dbo:` prefix for DBpedia, we perform the comparative evaluation using only the Wikidata version of the dataset. The evaluation is conducted using our best-performing LLM, Qwen2.5 7B, and the F1 Macro metric (Equation 5). As shown in Table 4, our system's performance is lower than SPINACH [15] and QAnswer [31]. The primary advantage of SPINACH comes from its use of a state-of-the-art proprietary LLM, GPT-4o. Meanwhile, QAnswer relies on pre-defined SPARQL templates, which help minimize syntactic or logical errors. However, this template-based approach lacks the flexibility to handle more complex or out-of-distribution queries. In contrast, our system generates SPARQL queries dynamically using an LLM, allowing for greater generalization across a wider range of questions and making it more adaptable to open-domain scenarios. The trade-off, however, is that generative methods are inherently more prone to hallucinations and incorrect query formulations, which can negatively impact overall performance.

| System | F1 Macro |
|---|---|
| **SPINACH GPT-4o [15]** | **0.746** |
| QAnswer [31] | 0.446 |
| **FrOG with Qwen2.5 7B Instruct (ours)** | **0.329** |
| Platypus [32] | 0.150 |
| DeepPavlov [33] | 0.124 |

**Table 4**
Benchmarking Comparison

## 5.3. Ablation Study

We evaluate the impact of key components in our pipeline—verbalization, chain-of-thought (CoT), few-shot examples, and vector-based ontology retrieval—by systematically removing each component to measure its contribution to overall performance. The experiments are conducted using the best-performing model for each knowledge base, Qwen2.5 7B. A more detailed analysis of the ablation study can be found in our technical report [30].

### 5.3.1. Verbalization

| Configuration | Knowledge Base | Jaccard Similarity |
|---|---|---|
| **Qwen2.5 7B** | **Wikidata** | **0.458** |
| Qwen2.5 7B w/o Verbalization | Wikidata | 0.334 |
| **Qwen2.5 7B** | **DBpedia** | **0.517** |
| Qwen2.5 7B w/o Verbalization | DBpedia | 0.516 |
| Qwen2.5 7B | Curriculum KG | 0.805 |
| **Qwen2.5 7B w/o Verbalization** | **Curriculum KG** | **0.949** |

**Table 5**
Ablation Study Results on Verbalization

Table 5 shows the impact of verbalization on system performance. Its removal led to a significant performance drop on Wikidata (0.458 ⟶ 0.334) and a minor decrease on DBpedia (0.517 ⟶ 0.516). While the drop on DBpedia is minor, verbalization proves essential for complex KGs, aiding LLMs in interpreting simple natural language queries. Without it, models struggle with single-hop queries. By converting extracted triples into readable sentences, verbalization ensures better semantic alignment for accurate query generation.

Interestingly, omitting verbalization improved accuracy on the curriculum KG (0.805 ⟶ 0.949). This suggests that verbalization acted as a bottleneck, limiting aggregation-based queries and failing to retrieve effective templates. For simpler, structured KGs, direct SPARQL query generation proved more effective.

These findings indicate that verbalization should be selectively applied, benefiting complex KGs but introducing redundancy for domain-specific datasets with straightforward queries.

### 5.3.2. Chain-of-Thought (CoT)

| Configuration | Knowledge Base | Jaccard Similarity |
|---|---|---|
| **Qwen2.5 7B** | **Wikidata** | **0.458** |
| Qwen2.5 7B w/o CoT | Wikidata | 0.436 |
| **Qwen2.5 7B** | **DBpedia** | **0.517** |
| Qwen2.5 7B w/o CoT | DBpedia | 0.455 |
| Qwen2.5 7B | Curriculum KG | 0.805 |
| Qwen2.5 7B w/o Verbalization | Curriculum KG | 0.949 |
| Qwen2.5 7B w/o CoT | Curriculum KG | 0.808 |
| **Qwen2.5 7B w/o Verbalization & CoT** | **Curriculum KG** | **0.976** |

**Table 6**
Ablation Study Results on CoT

Table 6 shows the impact of removing CoT across datasets. For Wikidata and DBpedia, excluding CoT resulted in a performance drop (0.458 → 0.436 and 0.517 → 0.455, respectively), indicating its role in improving SPARQL query generation. While helpful, CoT is not essential, mainly benefiting complex or open-ended queries.

In contrast, for the Curriculum KG, removing CoT had minimal impact or even improved results (with verbalization: 0.805 → 0.808, without verbalization: 0.949 → 0.976). This suggests CoT introduces unnecessary complexity in simpler datasets with direct queries, where intermediate reasoning is redundant.

These findings emphasize that CoT's effectiveness is dataset-dependent. For structured KGs like Wikidata and DBpedia, CoT enhances reasoning and query formulation, while for simpler datasets like the Curriculum KG, omitting CoT can streamline query generation and improve efficiency.

### 5.3.3. Few Shots

| Configuration | Knowledge Base | Jaccard Similarity |
|---|---|---|
| **Qwen2.5 7B** | **Wikidata** | **0.458** |
| Qwen2.5 7B w/o Few Shots | Wikidata | 0.342 |
| **Qwen2.5 7B** | **DBpedia** | **0.517** |
| Qwen2.5 7B w/o Few Shots | DBpedia | 0.410 |
| Qwen2.5 7B | Curriculum KG | 0.805 |
| **Qwen2.5 7B w/o Verbalization** | **Curriculum KG** | **0.949** |
| Qwen2.5 7B w/o Few Shots | Curriculum KG | 0.724 |
| Qwen2.5 7B w/o Verbalization & Few Shots | Curriculum KG | 0.651 |

**Table 7**
Ablation Study Results on Few Shots

Table 7 demonstrates the significant impact of few-shot learning on SPARQL query generation. Removing few shots caused notable performance drops across all datasets: Wikidata (0.458 → 0.342), DBpedia (0.517 → 0.410), and Curriculum KG (0.805 → 0.724). The effect was even more pronounced in Curriculum KG when verbalization was also removed (0.949 → 0.651).

Few shots guide the model in structuring SPARQL queries by providing dataset-specific examples. They help align model outputs with dataset constraints, such as ensuring URIs are returned in QALD-9-Plus or handling custom entity relationships in Curriculum KG. This is especially crucial for domain-specific datasets, where pre-trained knowledge alone is insufficient to generate accurate queries.

For general datasets like Wikidata and DBpedia, few shots refine query accuracy. For custom datasets like Curriculum KG, few shots are essential for adapting to unique ontologies. The results underscore the necessity of few-shot learning in bridging the gap between generic LLM capabilities and dataset-specific query requirements.

### 5.3.4. Ontology Retrieval

| Configuration | Knowledge Base | Jaccard Similarity |
|---|---|---|
| **Qwen2.5 7B** | **Wikidata** | **0.458** |
| Qwen2.5 7B w/o Ontology Retrieval | Wikidata | 0.377 |
| **Qwen2.5 7B** | **DBpedia** | **0.517** |
| Qwen2.5 7B w/o Ontology Retrieval | DBpedia | 0.381 |
| Qwen2.5 7B | Curriculum KG | 0.805 |
| **Qwen2.5 7B w/o Verbalization** | **Curriculum KG** | **0.949** |
| Qwen2.5 7B w/o Ontology Retrieval | Curriculum KG | 0.183 |
| Qwen2.5 7B w/o Verbalization & Ontology Retrieval | Curriculum KG | 0.000 |

**Table 8**
Ablation Study Results on Ontology Retrieval

Table 8 highlights the crucial role of ontology (class and property) retrieval in SPARQL query generation. Removing this component led to significant performance declines across all datasets: Wikidata (0.458 → 0.377), DBpedia (0.517 → 0.381), and Curriculum KG (0.805 → 0.183). Notably, the Curriculum KG score plummeted to 0.000 when both verbalization and ontology retrieval were removed, indicating the model's complete inability to generate meaningful queries.

Ontology retrieval is essential for identifying relevant classes and properties, ensuring accurate query construction. The drastic impact on Curriculum KG underscores its necessity for domain-specific datasets, where the LLM lacks prior exposure to the underlying ontology. In contrast, Wikidata and DBpedia, which the model may have partially encountered during training, showed smaller but still significant declines, emphasizing the retrieval step's importance in refining ontological alignment.

Interestingly, the Curriculum KG score remained at 0.183 when verbalization was retained, suggesting that verbalization can partially mitigate retrieval loss by inferring class and property relationships in natural language. However, this compensation is limited, as verbalization alone cannot handle complex queries that require precise class and property relationships.

#### 5.3.5. Ablation Study Summary

Ontology retrieval is the most critical component for query accuracy, particularly for domain-specific datasets. Without it, the model struggles to identify relevant ontology elements, leading to severe performance degradation. Even for open KGs like DBpedia and Wikidata, where the LLM has some prior knowledge, retrieval remains indispensable for precise query generation.

### 5.4. Inference Time Evaluation

We evaluated the inference time of the proposed pipeline on 5 queries per knowledge base from the Indonesian test set, using the default configuration. Table 9 reports the average times across components in seconds.

| Component | Wikidata | DBpedia | Curriculum KG |
|---|---|---|---|
| Translation | 0.06 s | 0.07 s | 0.068 s |
| Entity Linking | 9.11 s | 14.31 s | 7.104 s |
| Verbalization | 4.45 s | 2.64 s | 0.20 s |
| Ontology Retrieval | 9.97 s | 26.30 s | 29.98 s |
| SPARQL Generation | 37.42 s | 35.94 s | 28.60 s |
| Answer Generation | 2.18 s | 2.31 s | 2.90 s |
| **Total Avg. Inference Time** | **45.35 s** | **54.20 s** | **44.67 s** |
| Require SPARQL Generation Queries Avg. Time | 63.01 s | 77.41 s | 67.55 s |
| Verbalization Only Queries Avg. Time | 18.86 s | 19.39 s | 10.37 s |

**Table 9**
Average inference time per query and per component across knowledge bases

Overall, Curriculum KG achieved the fastest inference times due to its smaller and simpler ontology. Wikidata and DBpedia produced comparable results, although DBpedia required more time mainly due to slower ontology retrieval.

## 6. Conclusions and Future Work

In this research, we proposed FrOG, a framework of open GraphRAG. The framework consists of several key components: translation, entity linking, retrieval, and answer generation. In the retrieval stage, we differentiate between simple (single-hop) and complex questions. For simple questions, we leverage verbalization-based text retrieval to obtain relevant information, whereas for complex questions, we integrate SPARQL query generation using an LLM.

Our experiments demonstrate that Qwen2.5 7B achieves the best performance across all datasets using the optimal configuration identified through ablation analysis: Wikidata (0.458), DBpedia (0.517), and the Curriculum KG (0.976). Notably, simpler ontologies, such as the Curriculum KG, yield higher performance. Furthermore, the multilingual capability of the system was tested with Indonesian queries, achieving an accuracy of 0.362. Although lower than its English counterpart, this result highlights the system's ability to process multilingual queries through translation and retrieval mechanisms.

Despite moderate performance on Wikidata and DBpedia, we believe potential accuracy could be further improved, given the presence of inaccuracies within the dataset (see Appendix A). The ablation study highlights ontology (class and property) retrieval as the most critical component, with its removal causing substantial accuracy drops, particularly reducing curriculum KG performance to

zero. Interestingly, for simpler KGs, verbalization negatively impacts performance due to question misinterpretation and difficulties handling aggregation and complex operations.

Finally, this research successfully addresses the research questions by:

1. Developing FrOG, a RAG system that utilizes KGs as the primary knowledge base, with its pipeline architecture detailed in Section 3.
2. Identifying Qwen2.5 7B as the best-performing LLM, as demonstrated in Section 5.1.1.
3. Determining the most influential component within the architecture (i.e., "class and property retrieval" component) through the ablation study (cf. Section 5.3).

**Future Work.** Future work in this research aims to address several limitations and lead to significant improvements. One promising direction is to fine-tune LLMs for SPARQL query generation, which would enhance their ability to understand query structures, entity relationships, and natural language mappings, resulting in more precise query generation. Another avenue is to develop a language-native pipeline, particularly for Indonesian, as the current system solely relies on translation, which may introduce inaccuracies. A dedicated pipeline for Indonesian would eliminate translation errors and better handle linguistic nuances, further improving overall performance.

We also aim to explore the use of larger LLMs to boost query generation performance, with a focus on investigating how model size interacts with task complexity or the type of knowledge base. A deeper analysis of how parameter count influences performance will help in identifying the optimal model size for a particular use case while addressing computational efficiency and resource constraints. Another potential improvement involves introducing a classifier for verbalization applicability, such as intent classification, to determine whether a question is best handled using verbalization. This would optimize the query generation process by applying alternative strategies for complex queries. We further realize that scaling the system to larger knowledge graphs is essential, as the current implementation is limited by `rdflib`'s in-memory storage. Related to this, future work could explore several persistent storage solutions, such as Apache Jena, GraphDB, or Virtuoso, to enhance scalability and efficiency in supporting larger datasets. Additionally, we believe that extending this work beyond text-based KGs would be an interesting direction, as many real-world KGs now incorporate multimodal information. Finally, expanding the evaluation to include user studies and latency benchmarks will provide deeper insights into the practical usability of this system.

## Acknowledgments

This research was supported by the Wikidata Research Grant 2024 from Wikimedia Indonesia, whose funding and commitment to open knowledge made this study possible. The work of Fajar J. Ekaputra is supported by the Austrian Science Fund (FWF) Bilateral AI (Grant Nr. 10.55776/COE12) and Horizon Europe PERKS (Grant Nr. 101120323).

## Declaration on Generative AI

During the preparation of this work, the author(s) used GPT-4o in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

## References

[1] S. Dai, Y. Zhou, L. Pang, W. Liu, X. Hu, Y. Liu, X. Zhang, G. Wang, J. Xu, Neural retrievers are biased towards llm-generated content, in: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '24, Association for Computing Machinery, New

York, NY, USA, 2024, p. 526–537. URL: https://remote-lib.ui.ac.id:2075/10.1145/3637528.3671882. doi:10.1145/3637528.3671882.

[2] A. Roberts, C. Raffel, N. Shazeer, How much knowledge can you pack into the parameters of a language model?, 2020. URL: https://arxiv.org/abs/2002.08910. arXiv:2002.08910.

[3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive nlp tasks, in: Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20, Curran Associates Inc., Red Hook, NY, USA, 2020.

[4] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, W. tau Yih, Dense passage retrieval for open-domain question answering, 2020. URL: https://arxiv.org/abs/2004.04906. arXiv:2004.04906.

[5] B. Sarmah, B. Hall, R. Rao, S. Patel, S. Pasquali, D. Mehta, Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction, 2024. URL: https://arxiv.org/abs/2408.04948. arXiv:2408.04948.

[6] H. Abu-Rasheed, C. Weber, J. Zenkert, M. Dornhöfer, M. Fathi, Transferrable framework based on knowledge graphs for generating explainable results in domain-specific, intelligent information retrieval, Informatics 9 (2022). URL: https://www.mdpi.com/2227-9709/9/1/6. doi:10.3390/informatics9010006.

[7] B. Peng, Y. Zhu, Y. Liu, X. Bo, H. Shi, C. Hong, Y. Zhang, S. Tang, Graph retrieval-augmented generation: A survey, 2024. URL: https://arxiv.org/abs/2408.08921. arXiv:2408.08921.

[8] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, 2019. URL: https://arxiv.org/abs/1907.11692. arXiv:1907.11692.

[9] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, J. Larson, From local to global: A graph rag approach to query-focused summarization, 2024. URL: https://arxiv.org/abs/2404.16130. arXiv:2404.16130.

[10] Y. Cao, Z. Gao, Z. Li, X. Xie, K. Zhou, J. Xu, Lego-graphrag: Modularizing graph-based retrieval-augmented generation for design space exploration, 2025. URL: https://arxiv.org/abs/2411.05844. arXiv:2411.05844.

[11] M. R. A. H. Rony, U. Kumar, R. Teucher, L. Kovriguina, J. Lehmann, Sgpt: A generative approach for sparql query generation from natural language questions, IEEE Access 10 (2022) 70712–70723. doi:10.1109/ACCESS.2022.3188714.

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, Language models are unsupervised multitask learners, OpenAI Blog (2019). URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[13] V. Emonet, J. Bolleman, S. Duvaud, T. M. de Farias, A. C. Sima, Llm-based sparql query generation from natural language over federated knowledge graphs, 2024. URL: https://arxiv.org/abs/2410.06062. arXiv:2410.06062.

[14] L. Kovriguina, R. Teucher, D. Radyush, D. Mouromtsev, Sparqlgen: One-shot prompt-based approach for sparql query generation, in: International Conference on Semantic Systems, 2023. URL: https://api.semanticscholar.org/CorpusID:265309659.

[15] S. Liu, S. J. Semnani, H. Triedman, J. Xu, I. D. Zhao, M. S. Lam, Spinach: Sparql-based information navigation for challenging real-world questions, 2024. URL: https://arxiv.org/abs/2407.11417. arXiv:2407.11417.

[16] J. Ongris, E. Tjitrahardja, F. Darari, F. Ekaputra, Towards an Open NLI LLM-based System for KGs: A Case Study of Wikidata, in: The 7th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), 2024.

[17] H. Jiang, Q. Wu, X. Luo, D. Li, C.-Y. Lin, Y. Yang, L. Qiu, LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression, in: L.-W. Ku, A. Martins, V. Srikumar (Eds.), Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Bangkok, Thailand, 2024, pp. 1658–1677. URL: https://aclanthology.org/2024.acl-long.91. doi:10.18653/v1/2024.acl-long.91.

[18] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Commun. ACM 57 (2014) 78–85. URL: https://doi.org/10.1145/2629489. doi:10.1145/2629489.
[19] M. M. Cantallops, S. Sánchez-Alonso, E. García-Barriocanal, A systematic literature review on wikidata, Data Technol. Appl. 53 (2019) 250–268. URL: https://api.semanticscholar.org/CorpusID:202036639.
[20] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, S. Hellmann, Dbpedia - a crystallization point for the web of data, Journal of Web Semantics 7 (2009) 154–165. URL: https://www.sciencedirect.com/science/article/pii/S1570826809000225. doi:https://doi.org/10.1016/j.websem.2009.07.002, the Web of Data.
[21] D. C. Mandolang, curriculum-knowledge-graph, https://github.com/danielcm585/curriculum-knowledge-graph, 2024.
[22] U. Zaratiana, N. Tomeh, P. Holat, T. Charnois, Gliner: Generalist model for named entity recognition using bidirectional transformer, 2023. URL: https://arxiv.org/abs/2311.08526. arXiv:2311.08526.
[23] A. Perevalov, D. Diefenbach, R. Usbeck, A. Both, Qald-9-plus: A multilingual dataset for question answering over dbpedia and wikidata translated by native speakers, in: 2022 IEEE 16th International Conference on Semantic Computing (ICSC), 2022, pp. 229–234. doi:10.1109/ICSC52841.2022.00045.
[24] K. Song, X. Tan, T. Qin, J. Lu, T.-Y. Liu, Mpnet: masked and permuted pre-training for language understanding, in: Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20, Curran Associates Inc., Red Hook, NY, USA, 2020.
[25] A. Both, A. Perevalov, Towards llm-driven natural language generation based on sparql queries and rdf knowledge graphs, in: TEXT2KG/DQMLKG@ESWC, 2024. URL: https://api.semanticscholar.org/CorpusID:271963651.
[26] Mistral AI, Mistral nemo, 2024. URL: https://mistral.ai/news/mistral-nemo.
[27] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, et al., The llama 3 herd of models, 2024. URL: https://arxiv.org/abs/2407.21783. arXiv:2407.21783.
[28] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, K. Dang, Y. Fan, Y. Zhang, A. Yang, R. Men, F. Huang, B. Zheng, Y. Miao, S. Quan, Y. Feng, X. Ren, X. Ren, J. Zhou, J. Lin, Qwen2.5-coder technical report, 2024. URL: https://arxiv.org/abs/2409.12186. arXiv:2409.12186.
[29] Qwen Team, Qwen2.5: A party of foundation models, https://qwenlm.github.io/blog/qwen2.5/, 2024. Accessed: 2024-12-05.
[30] J. Ongris, E. Tjitrahardja, Frog: Framework of open graphrag, 2025. URL: https://juang.id/theses/2025_Ongris_Tjitrahardja.pdf.
[31] D. Diefenbach, A. Both, K. Singh, P. Maret, Towards a question answering system over the semantic web, 2018. URL: https://arxiv.org/abs/1803.00832. arXiv:1803.00832.
[32] T. Pellissier Tanon, M. D. de Assunção, E. Caron, F. M. Suchanek, Demoing platypus – a multilingual question answering platform for wikidata, in: A. Gangemi, A. L. Gentile, A. G. Nuzzolese, S. Rudolph, M. Maleshkova, H. Paulheim, J. Z. Pan, M. Alam (Eds.), The Semantic Web: ESWC 2018 Satellite Events, Springer International Publishing, Cham, 2018, pp. 111–116.
[33] M. Burtsev, A. Seliverstov, R. Airapetyan, M. Arkhipov, D. Baymurzina, N. Bushkov, O. Gureenkova, T. Khakhulin, Y. Kuratov, D. Kuznetsov, A. Litinsky, V. Logacheva, A. Lymar, V. Malykh, M. Petrov, V. Polulyakh, L. Pugachev, A. Sorokin, M. Vikhreva, M. Zaynutdinov, DeepPavlov: Open-source library for dialogue systems, in: F. Liu, T. Solorio (Eds.), Proceedings of ACL 2018, System Demonstrations, Association for Computational Linguistics, Melbourne, Australia, 2018, pp. 122–127. URL: https://aclanthology.org/P18-4021/. doi:10.18653/v1/P18-4021.

## A. Appendix: Dataset Issues

During our experiments, we identified several dataset inconsistencies, particularly in the QALD-9-Plus dataset:

1. **Inaccurate Ground Truth (1):** The question "How many languages are spoken in Colombia?" was associated with an incorrect ground truth SPARQL query: `SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { ?uri rdf:type dbo:Language . dbr:Colombia dbo:language ?uri }`. However, there was no direct connection between `dbr:Colombia` and `dbo:language`, leading to a result of 0, which is incorrect. Our system correctly generated the query: `SELECT COUNT(DISTINCT ?lang) WHERE { ?lang dbo:spokenIn dbr:Colombia }`, which returned 97, the correct answer. However, since the ground truth was incorrect, our correct answer was misclassified as wrong.
2. **Inaccurate Ground Truth (2):** The question "How many films did Hal Roach produce¿' was linked to an incomplete ground truth query: `SELECT (COUNT(?uri) as ?c) WHERE { ?uri wdt:P162 wd:Q72792 . }`. This query failed to restrict results to instances of films. In contrast, our system generated: `SELECT (COUNT(DISTINCT ?uri) AS ?count) WHERE { ?uri wdt:P31 wd:Q11424 ; wdt:P162 wd:Q72792. }`, which correctly filters for entities classified as films (`wd:Q11424`). However, due to the ground truth's lack of specificity, our accurate result was marked incorrect.
3. **Ambiguous Question (1):** The question "How big is the total area of North Rhine-Westphalia?" was ambiguous regarding the expected unit of measurement. The ground truth query retrieved the area in square meters: `SELECT ?tarea WHERE { dbr:North_Rhine-Westphalia dbo:areaTotal ?tarea }`. However, our system retrieved `dbo:PopulatedPlace/areaTotal`, which provides the value in square kilometers. While both answers were correct, the unit difference led to our result being classified as incorrect.
4. **Ambiguous Question (2):** The question "Give me all soccer clubs in Spain." was also ambiguous. Our system generated: `SELECT DISTINCT ?uri WHERE { ?uri a dbo:SoccerClub ; dbo:country dbr:Spain }`, which is semantically correct. However, the ground truth query: `SELECT DISTINCT ?uri WHERE { ?uri a dbo:SoccerClub { ?uri dbo:ground dbr:Spain } UNION { ?uri <http://dbpedia.org/property/ground> ?ground FILTER regex(?ground, "Spain") } }`, applied additional constraints on stadium locations, making it overly specific. As a result, our correct query was misclassified as incorrect.