Information-Needs-Guided Virtual Knowledge Graph Enrichment via Large Language Models

Lin Ren[1,2], **Guohui Xiao**[1,2,*], **Guilin Qi**[1,2], **Wenjie Du**[1,2], **Yishuai Geng**[1,2], **Haohan Xue**[1,2], **Zhiyan Yue**[1,2], **Mingxuan Li**[3], **Marco Di Panfilo**[4], **Davide Lanti**[4], **Kamal Hamaz**[4], **Linfang Ding**[5]

[1]School of Computer Science and Engineering, Southeast University, Nanjing, China
[2]Key Laboratory of New Generation Artificial Intelligence Technology and Its Interdisciplinary Applications (Southeast University), Ministry of Education, China
[3]Dalian University of Technology, Liaoning, China
[4]Free University of Bozen-Bolzano, Italy
[5]Jiangsu Key Laboratory of Soil and Water Processes in Watershed, College of Geography and Remote Sensing, Hohai University, Nanjing, China
{renlin, guohui.xiao, gqi, wenjiedu, ysgeng, thex1ay, yuezhiyan}@seu.edu.cn, lmx71017@mail.dlut.edu.cn, {mdipanfilo, davide.lanti, kamal.hamaz}@unibz.it, linfang.ding@hhu.edu.cn

Abstract

Virtual Knowledge Graphs (VKGs) provide an effective solution for data integration by mapping heterogeneous data sources to a unified ontology. However, existing VKG construction frameworks primarily focus on one-shot construction, which often results in partial data coverage and support for only initial information needs. After the VKG has been deployed, new information needs will inevitably arise over time. Therefore, enriching VKGs to support evolving information needs remains an expert-intensive iterative task. In this work, we formulate the task of Information-Needs-Guided VKG Enrichment (IN-VKGE), and propose an iterative framework that leverages large language models to assess whether information needs can be supported using SPARQL execution feedback and generate ontology and mapping enrichment proposals. Experiments on two real-world VKGs show that our approach outperforms existing paradigms, and produces enrichment proposals that receive high expert ratings for effectively resolving the identified information needs.

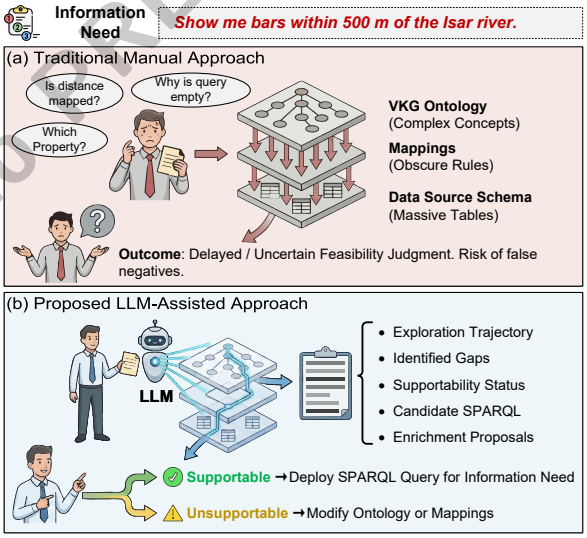

Figure 1: Traditional manual vs. our LLM-assisted approach for VKG enrichment support.

1 Introduction

Virtual Knowledge Graphs (VKGs), also known as Ontology-Based Data Access (OBDA), provide a unified ontological view over diverse relational databases, enabling semantic queries without copying the data. This paradigm allows accessing data through a conceptual layer, a foundational principle of VKG [Calvanese *et al.*, 2007; Lenzerini, 2011; Daraio *et al.*, 2016; Xiao *et al.*, 2018; Cima *et al.*, 2023]. Their capability to support queries over up-to-date data with minimal storage overhead has led to widespread adoption in industry, supported by robust VKG systems [De Giacomo *et al.*, 2012; Rodriguez-Muro *et al.*, 2013; Sequeda and Miranker, 2017; Xiao *et al.*, 2019; Sequeda *et al.*, 2023]. VKG construction is rarely a one-shot effort; instead, it evolves iteratively as *information needs*, the data requirements expressed in natural language by domain users, emerge and change over the system's lifecycle [Hovland *et al.*, 2017]. However, assessing whether a real-world information need can be supported by a VKG specification remains time-consuming and expert-intensive. This process requires domain experts to deeply understand the ontology structure and mapping rules,

and to craft executable SPARQL queries. However, when these queries return no results, the cause is inherently ambiguous, as identical empty results may stem from ontology misalignment, missing mappings, or unrealizable SQL joins. As VKG specifications grow in scale and complexity, this process becomes increasingly challenging.

Existing works mainly follow a *schema-centric, one-shot VKG construction* paradigm. Tools such as Ontop [Calvanese *et al.*, 2017], D2RQ [Bizer and Seaborne, 2004], and BootOX [Jiménez-Ruiz *et al.*, 2015] focus on automating mapping generation directly from database schemas, occasionally enriched by schema alignment techniques [Jiménez-Ruiz *et al.*, 2015; de Medeiros *et al.*, 2015]. Recent works such as LLM4VKG [Xiao *et al.*, 2025] also operate under this paradigm. However, addressing information needs is a fundamental principle of VKG design [Hovland *et al.*, 2017], so the aforementioned paradigm presents three key limitations: *(1)* it lacks the mechanisms to identify which information needs remain unsupported after construction, meaning experts have to manually investigate coverage gaps; *(2)* when gaps are detected, experts must independently determine how to extend mappings or ontology vocabulary; *(3)* one-shot VKG construction is rarely sufficient; real-world applications inevitably encounter new information needs during deployment, requiring iterative enrichment. These limitations motivate an information-needs-guided approach that proactively surfaces coverage gaps and proposes targeted enrichments.

Large Language Models (LLMs) [DeepSeek-AI, 2024; OpenAI, 2025; Google, 2025; Anthropic, 2025], with their ability to process and interpret complex specifications [Chen, 2024; Azaria *et al.*, 2024; Luo *et al.*, 2024], offer a promising way to address these challenges. As demonstrated by Xiao *et al.* [2025] for VKG construction, LLMs can effectively handle the complexity of ontology-schema relationships. Inspired by this, in this work, we aim to explore the potential of LLMs to assist with VKG enrichment. We formulate the task of *Information-Needs-Guided VKG Enrichment (IN-VKGE)*: given a VKG specification and an information need, we assess whether the need can be supported and generate enrichment proposals for the given VKG. To this end, we propose an LLM-based framework that iteratively explores VKG specifications and generates enrichment proposals. Figure 1 contrasts the traditional manual approach with our proposed LLM-assisted approach. In the traditional manual approach, domain experts must navigate complex VKG ontologies, obscure mapping rules, and massive data source schemas. In contrast, our proposed framework leverages LLMs to automatically explore the VKG specification, generate candidate SPARQL queries, and provide structured enrichment proposals to guide expert decision-making.

Our contributions are threefold: *(1)* we formulate a new task, IN-VKGE, which uses information needs to guide VKG enrichment; *(2)* we propose the first framework for IN-VKGE, featuring memory-driven iteration and supportability assessment for information-needs-guided VKG enrichment via LLMs; *(3)* we construct the first IN-VKGE dataset and demonstrate the effectiveness of the proposed framework through experimentation. All code and data are available at https://github.com/HomuraT/IN-VKGEnrichment.

2 Related Work

2.1 VKG Construction and Maintenance

The lifecycle of a VKG encompasses not only its initial construction but also its continuous evolution to meet changing information needs. Existing tooling has largely focused on *schema-centric construction*. Systems like Ontop [Calvanese *et al.*, 2017], D2RQ [Bizer and Seaborne, 2004], and MIRROR [de Medeiros *et al.*, 2015] automate the generation of initial mappings from relational schemas. Tools such as BootOX [Jiménez-Ruiz *et al.*, 2015] and COMA++ [Aumueller *et al.*, 2005] further support this by leveraging schema constraints for ontology alignment. Recently, LLM4VKG [Xiao *et al.*, 2025] utilized LLMs to enhance this setup phase but remains fundamentally *needs-agnostic*. However, these approaches prioritize structure preservation over information needs, often resulting in generic VKGs that lack the specific semantics needed to answer domain questions and do not address the diagnosis of gaps for evolving, post-deployment needs. In contrast, *needs-driven evolution* prioritizes the fulfillment of user requirements, driving the VKG specification to evolve iteratively as new information needs emerge. Methodologies like *pay-as-you-go* [Pinkel *et al.*, 2013; Sequeda and Miranker, 2017] and industrial experiences [Kharlamov *et al.*, 2015; Hovland *et al.*, 2017] emphasize iteratively enriching the VKG based on actual business questions. This paradigm ensures the system remains relevant to information needs. Visual query systems [Soylu *et al.*, 2014] have also been proposed to help non-technical users express these needs. Despite the importance of this iterative process, automated support for *VKG enrichment* is lacking. Our work bridges this gap by leveraging LLMs to automate the generation of enrichment proposals to support expert decision-making.

2.2 Natural Language Interface to KG

Current approaches for accessing knowledge graphs via natural language largely fall into two paradigms: *(1) Semantic Parsing.* This paradigm focuses on mapping natural language to logical forms like SPARQL [Berant *et al.*, 2013; Soru *et al.*, 2017; Herzig and Berant, 2017]. Recent advancements enhance this by fine-tuning LLMs [Zhang *et al.*, 2024] or employing retrieval-augmented generation (RAG) to inject schema context [Ma *et al.*, 2025a; Gao *et al.*, 2023]. Notable among these, ReAct-based approaches [Yao *et al.*, 2023; Brei *et al.*, 2025] integrate reasoning and acting in an iterative loop, enabling LLMs to explore knowledge graph schemas and revise SPARQL queries through trial-and-error. *(2) Direct Reasoning.* This paradigm bypasses explicit query generation in favor of direct reasoning over graph structures. Methods such as iterative graph traversal approaches [Luo *et al.*, 2024; Sun *et al.*, 2024; Ma *et al.*, 2025b] navigate the entity space to infer answers. Recent works further empower LLMs with graph reasoning capabilities through structured prompting [Wang *et al.*, 2024; Jiang *et al.*, 2023], tool augmentation [Zhang, 2023], and ensemble reasoning for decision-making over semantic knowledge graphs [Rashid and McGuinness, 2025]. These methods assume that the information need is answerable by the cur-

rent KG. They lack the capability to detect coverage gaps or propose enrichments when the requisite data or mappings are missing from the VKG specification. Our work bridges this gap by formulating VKG enrichment guided by information needs, which generates executable SPARQL queries to assess VKG coverage and provides enrichment proposals to support expert decision-making.

3 Preliminaries

3.1 Virtual Knowledge Graph

A *Virtual Knowledge Graph (VKG)* provides a unified semantic interface over relational databases without materializing RDF triples [Calvanese *et al.*, 2017]. A *VKG specification* is a triple $(\mathcal{T}, \mathcal{M}, \Sigma)$, where: *(1)* Σ denotes a *DB schema*, consisting of table schemata and database constraints (e.g., primary and foreign keys). *(2)* $\mathcal{T}$ denotes an *ontology* (TBox), a set of axioms expressed in terms of *class names* (NC), *object property names* (NP), and *data property names* (ND). *(3)* $\mathcal{M}$ denotes a set of *mappings*, each connecting a SQL query over Σ to ontology concepts (i.e., classes and properties) in $\mathcal{T}$, enabling on-the-fly translation of SPARQL queries to SQL.

VKGs expose a *virtual ABox* $\mathcal{M}(\mathcal{D})$ that is computed dynamically from the database instance $\mathcal{D}$ through mappings, ensuring data freshness while supporting semantic querying. Formal definitions of mappings and virtual ABox semantics are provided in Appendix B.

3.2 Problem Definition

We formulate the *Information-Needs-Guided VKG Enrichment (IN-VKGE)* task as follows: Given a VKG specification $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \Sigma)$, a database instance $\mathcal{D}$, and an information need q, where q is an informal natural language description of data requirements to be supported through SPARQL queries [Hovland *et al.*, 2017]. We assume $\mathcal{D}$ contains the necessary data, but q cannot be answered over $\mathcal{P}$ with $\mathcal{D}$ yet, possibly due to the absence of required mappings or ontology vocabulary. The goal of IN-VKGE is to: *(1)* generate a SPARQL query $\hat{\rho}$ that captures q, and *(2)* propose enrichments $(\mathcal{M}^*, \mathcal{T}^*)$, such that when executing $\hat{\rho}$ over $\mathcal{P}^* = (\mathcal{T} \cup \mathcal{T}^*, \mathcal{M} \cup \mathcal{M}^*, \Sigma)$ against $\mathcal{D}$, the returned results fulfill the information need q. Note that the quality of both the generated query $\hat{\rho}$ and the proposed enrichments $(\mathcal{M}^*, \mathcal{T}^*)$ typically requires expert evaluation.

4 Methodology

4.1 Framework Overview

The overall framework for the IN-VKGE task is illustrated in Figure 2. The framework comprises three main steps: *(1) VKG Vectorization*, which verbalizes and vectorizes VKG specifications into a vector database; *(2) Information-need–oriented Query Exploration*, which iteratively generates candidate SPARQL queries, executes them against the VKG endpoint, evaluates supportability, and produces the final query with enrichment proposals; and *(3) Expert Decision-Making & VKG Update*, which presents the exploration results and enrichment proposals to inform expert decision-making on VKG enrichment.

4.2 VKG Vectorization

Before exploring the supportability of information needs, we vectorize the VKG specification into a vector database. To enrich the semantic information of VKG elements, we employ a generative language model $\mathcal{L}_{gen}$ to verbalize each element (ontology terms and mapping rules). Both the original elements and their verbalizations are vectorized using an embedding model $\mathcal{L}_{emb}$. The vector database covers two indices $x \in \{onto, map\}$: *(1) Ontology Vocabulary* ($\mathcal{I}^{onto}$), containing vectors for classes and properties in $\mathcal{T}$; *(2) Mappings* ($\mathcal{I}^{map}$), containing vectors for R2RML mappings in $\mathcal{M}$, capturing the correspondence between Σ and $\mathcal{T}$. Let E_{onto} comprise classes and properties in $\mathcal{T}$, and E_{map} comprise mapping rules in $\mathcal{M}$. Each index $\mathcal{I}^x$ is constructed in two steps. First, each element is verbalized:

$$E_x^{verb} = \{\text{Verbalize}(e; \mathcal{L}_{gen}) \mid e \in E_x\}$$

Then, both the original elements and their verbalizations are vectorized:

$$\mathcal{I}^x = \{\text{Vectorize}(e; \mathcal{L}_{emb}) \mid e \in E_x \cup E_x^{verb}\}$$

For retrieval, the Retrieve function returns the top-k most relevant items as an ordered list, given a textual query τ:

$$\mathcal{C}_{rel}^x(\tau) = \text{Retrieve}(\tau, \mathcal{I}^x, k; \mathcal{L}_{emb})$$

where k is a fixed hyperparameter, and τ can be either the information need q or a refined retrieval target γ_i (detailed in Section 4.4). The constructed context aggregates results from both indices:

$$\mathcal{C}_{rel}(\tau) = \mathcal{C}_{rel}^{onto}(\tau) \cup \mathcal{C}_{rel}^{map}(\tau)$$

4.3 Information-need–oriented Query Exploration

Given the vectorized VKG, we can retrieve relevant context for the information-need-oriented query exploration. The exploration iteratively generates candidate SPARQL queries, executes them, and decides whether the information need can be supported, ultimately producing the final query and enrichment proposals.

Step 1: Candidate Generation

To explore whether an information need can be supported, we iteratively generate candidate SPARQL queries using the LLM $\mathcal{L}_{gen}$. At iteration i, a set $\hat{\boldsymbol{\rho}}_i$ of candidate SPARQL queries is generated via the LLM, conditioned on the information need q, the retrieved context $\mathcal{C}_{rel}(q)$, and the memory $\mathcal{B}_{i-1}$ from previous iterations (detailed in Section 4.4):

$$\hat{\boldsymbol{\rho}}_i = \text{GenerateCandidates}(q, \mathcal{C}_{rel}(q), \mathcal{B}_{i-1}; \mathcal{L}_{gen})$$

The generated candidates encompass various SPARQL query types for exploring the VKG.

Step 2: Execution and Result Inspection

Once the candidates $\hat{\boldsymbol{\rho}}_i$ are generated, they are executed against the VKG specification $\mathcal{P}$ and database instance $\mathcal{D}$:

$$R_i = \text{Execute}(\hat{\boldsymbol{\rho}}_i, \mathcal{P}, \mathcal{D})$$

where $R_i = \{(s^j, r^j) \mid \hat{\rho}_i^j \in \hat{\boldsymbol{\rho}}_i\}$ captures the execution status $s^j \in \{\textsc{Success}, \textsc{SyntaxError}, \textsc{Empty}\}$ and the corresponding result r^j (query results, error message, or empty set) for each candidate.

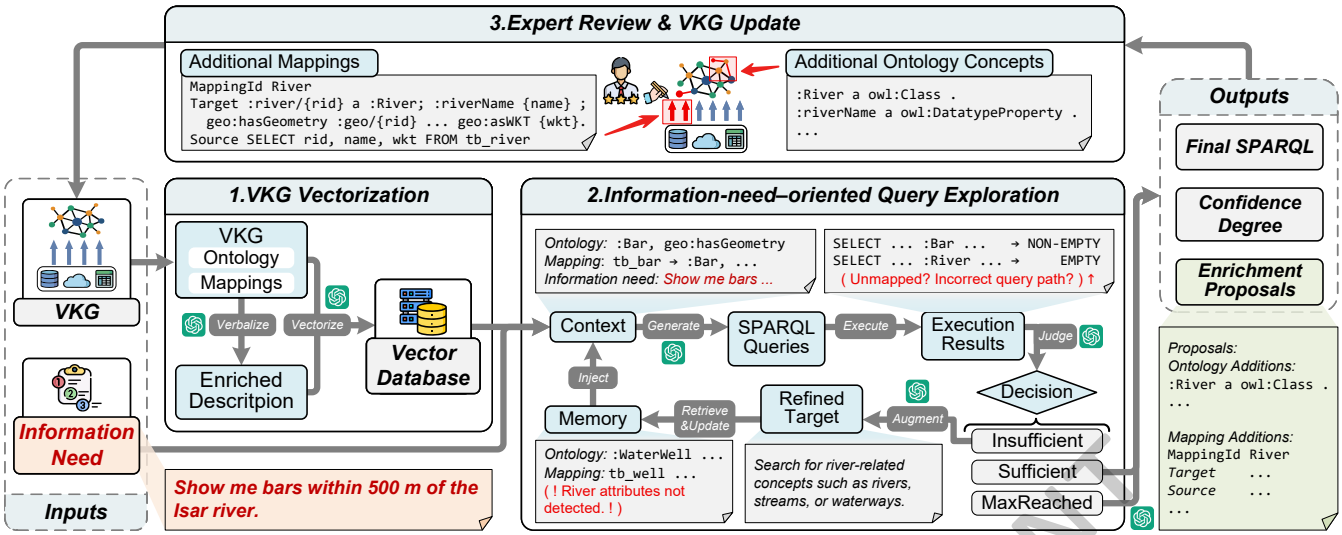

Figure 2: Overview of our framework. Operations involving the LLM icon denote the use a large language model.

Step 3: Decision and Proposal Generation

Based on the execution results, the next critical step is to determine if the information need has been met or if further exploration is required. We employ $\mathcal{L}_{gen}$ as a *judge* module (denoted Ψ) to analyze the execution results R_i and memory $\mathcal{B}_{i-1}$, producing a decision a_i and output o_i:

$$(a_i, o_i) = \Psi(q, \hat{\boldsymbol{\rho}}_i, R_i, \mathcal{B}_{i-1}; \mathcal{L}_{gen})$$

where $a_i \in \{\text{SUFFICIENT}, \text{INSUFFICIENT}, \text{MAXREACHED}\}$. The output o_i is defined as: *(1)* INSUFFICIENT: The execution results fail to adequately support the information need, yielding $o_i = \gamma_i$, where γ_i is a refined retrieval target to guide the next iteration (e.g., "find river-related concepts"); *(2)* SUFFICIENT: The execution results reliably support the information need, yielding $o_i = \hat{\rho}^*$, where $\hat{\rho}^* \in \hat{\boldsymbol{\rho}}_i$ is the final SPARQL query; *(3)* MAXREACHED: The iteration limit is reached, yielding $o_i = \hat{\rho}^\dagger$, where $\hat{\rho}^\dagger \notin \hat{\boldsymbol{\rho}}_i$ is a new SPARQL query generated based on q and $\mathcal{B}_i$ using $\mathcal{L}_{gen}$.

When the exploration terminates, the enrichment proposal $(\mathcal{M}^*, \mathcal{T}^*)$ is generated based on the memory bank $\mathcal{B}_i$:

$$(\mathcal{M}^*, \mathcal{T}^*) = \text{GenerateProposal}(q, \mathcal{B}_i, \mathcal{C}_{rel}(q); \mathcal{L}_{gen})$$

where $\mathcal{M}^*$ suggests mapping-level additions (e.g., new mappings to expose unmapped data), and $\mathcal{T}^*$ suggests ontology-level additions (e.g., adding missing properties or classes).

4.4 Memory Bank

To prevent repetitive errors, we maintain a *Memory Bank* $\mathcal{B}$ that records exploration traces across iterations. It guides subsequent candidate generation and decision-making based on queries and outcomes from previous iterations. At the end of each iteration i, the memory is updated with the candidate queries, their execution outcomes, and optionally the retrieved context based on γ_i:

$$\mathcal{B}_i = \mathcal{B}_{i-1} \cup \{(\hat{\boldsymbol{\rho}}_i, R_i, \mathcal{C}_{rel}(\gamma_i))\}$$

If no γ_i is generated (i.e., upon SUFFICIENT or MAXREACHED), $\mathcal{C}_{rel}(\gamma_i) = \emptyset$. In subsequent iterations, $\mathcal{B}_i$ guides candidate SPARQL generation based on queries and outcomes from previous iterations.

4.5 Expert Review and VKG Update

Following the query exploration process, the enrichment proposals are presented to the VKG experts. These outputs are designed to support expert decision-making.

Given the final SPARQL query $\hat{\rho} \in \{\hat{\rho}^*, \hat{\rho}^\dagger\}$ and the termination decision a_i, experts can: *(1)* SUFFICIENT: Directly adopt the query as a validated access pattern for the information need q; or *(2)* MAXREACHED: Review the exploration trajectory in $\mathcal{B}$ to understand why the need cannot be fully supported.

Based on the enrichment proposals $(\mathcal{M}^*, \mathcal{T}^*)$, experts can selectively apply suggested additions: *(1) Mapping Updates ($\mathcal{M}^*$)*: Add new R2RML mappings to enrich the VKG, enabling it to support the information need; and *(2) Ontology Updates ($\mathcal{T}^*$)*: Introduce missing classes or properties to enrich the ontology vocabulary and better align with information needs. Additionally, a textual report is generated to present the exploration process and suggested enrichments:

$$r = \text{GenerateReport}(q, \mathcal{B}_i, \mathcal{M}^*, \mathcal{T}^*; \mathcal{L}_{gen})$$

This expert-in-the-loop design ensures that VKG additions are validated by domain expertise before deployment.

In summary, given a VKG specification $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \Sigma)$ and a natural language information need q, the framework produces the output $(\hat{\rho}, \mathcal{M}^*, \mathcal{T}^*)$ along with a report r, enabling experts to efficiently review and update the VKG.

5 Datasets and Evaluation

5.1 Dataset Construction

To evaluate the IN-VKGE task, following the standard annotation protocols [Yu *et al.*, 2018; Dubey *et al.*, 2019], we

VKG	Ontology $\mathcal{T}$			Database $(\Sigma, \mathcal{D})$			#Sample
	|NC|	|NP|	|ND|	#Tab	#Col	#Row	
BGEE	32	23	6	6	29	97,474	82
NPD	342	142	238	70	962	257,271	116

Table 1: VKG dataset statistics.

constructed *manually annotated* datasets based on two real-world VKG scenarios: BGEE [Calvanese *et al.*, 2021] and NPD [Lanti *et al.*, 2015]. We selected these two scenarios to ensure diversity in both *domain* and *scale*. Further details (e.g., the annotation process) can be found in Appendix C.

Each annotated sample consists of *(1)* a natural-language information need q, and *(2)* an expert-written gold SPARQL query ρ that fulfills the information need q under the given VKG specification $\mathcal{P}$. After manual review and filtering, the sample counts and the VKG statistics are shown in Table 1.

5.2 Evaluation Metrics

We employ distinct evaluation strategies for each component.

Query Evaluation. Since different SPARQL formulations may be semantically equivalent, string-based comparison is insufficient. We evaluate query quality by *executing* both the gold query ρ and the generated query $\hat{\rho}$, and then comparing their results. We adopt two metrics: *Execution F1*, measuring answer correctness, and *Empty Rate*, measuring the proportion of queries returning empty results.

Enrichment Evaluation. For the enrichment proposals $(\mathcal{M}^*, \mathcal{T}^*)$, which lack ground truth, we conduct a human evaluation. VKG experts assess the proposals on three dimensions: *Identification*, *Validity*, and *Executability*.

Execution F1. We evaluate query correctness by comparing the *execution results* of the generated SPARQL against the gold SPARQL, which is robust to syntactic variations that yield equivalent answers. For the j-th sample, let G^j and P^j denote the gold and predicted answer tables, with C_g^j and C_p^j denoting respectively the number of columns in G^j and P^j. Let G_i^j (resp. P_i^j) be the multiset of values in the i-th column. The F1 score is computed in four steps:

Step 1: Column alignment. Following the Hungarian algorithm [Kuhn, 1955], we find a maximum-weight matching σ between gold columns and predicted columns that maximizes the total overlap $\sum_{i=1}^{C_g^j} |G_i^j \cap P_{\sigma(i)}^j|$, where $P_{\sigma(i)}^j = \emptyset$ for unmatched gold columns when $C_p^j < C_g^j$.

Step 2: Column-level F1. For each aligned column pair $(G_i^j, P_{\sigma(i)}^j)$, we compute:

$F1_{\text{col}}^{i,j} = 2 \cdot |G_i^j \cap P_{\sigma(i)}^j| / (|G_i^j| + |P_{\sigma(i)}^j|)$.

Step 3: Sample-level F1. We average the column F1 scores with a penalty for extra predicted columns:

$$F1_{\text{sample}}^j = \left(\frac{1}{C_g^j} \sum_{i=1}^{C_g^j} F1_{\text{col}}^{i,j} \right) \times \min(1, C_g^j / C_p^j).$$

Step 4: Dataset-level F1. The final F1 is the macro-average: $\text{F1} = \frac{1}{N} \sum_{j=1}^{N} F1_{\text{sample}}^j$.

6 Experiments

Our evaluation focuses on the two key aspects of IN-VKGE: *(1) Query Generation*: evaluating whether generated SPARQL queries $\hat{\rho}$ effectively capture the information need q; and *(2) Enrichment Proposal Generation*: assessing whether the termination decision a_i reliably distinguishes between supported and unsupported information needs, and the quality of the enrichment report r with proposals $(\mathcal{M}^*, \mathcal{T}^*)$.

6.1 Evaluation of Query Generation

IN-VKGE requires generating a SPARQL query $\hat{\rho}$ that captures the information need q. We evaluate query quality by comparing execution results against expert-written gold SPARQL queries ρ.

Comparative Paradigms

We compare against three representative paradigms from Text-to-SPARQL research: *(1) vanilla LLM* [Zahera *et al.*, 2024; Avila *et al.*, 2024; Allemang and Sequeda, 2024] provides the LLM with the ontology or mapping and the information needed for direct SPARQL generation. *(2) RAG* [Gao *et al.*, 2023; Emonet *et al.*, 2024; Ma *et al.*, 2025a] retrieves ontology or mapping fragments relevant to the information need for generation; *(3) ReAct* [Yao *et al.*, 2023; Jiang *et al.*, 2023; Brei *et al.*, 2025] interleaves reasoning and action steps (retrieving ontology/mappings, executing SPARQL) to explore the VKG based on the information need.

Implementation Details

In this study, we deploy VKGs with Ontop [Calvanese *et al.*, 2017][1], a state-of-the-art VKG system. All paradigms use identical LLM backends (*gpt-4o-mini* and *gpt-4o*, retrieval model (*qwen3-8b-embedding* , and evaluation metrics. For our framework, the retrieval parameter k is set to 10 for each of the two indices (ontology and mapping), with a maximum of 3 iterations. For RAG and ReAct, k is set to 50 to ensure sufficient context coverage; ReAct is configured with a maximum of 10 reasoning steps. All methods are evaluated on the original VKG; no enrichment proposals are applied. More implementation details are provided in Appendix D.

Query Generation Performance

Table 2 presents the comparative performance of query generation paradigms across two datasets (BGEE and NPD) using two LLM backends. As the results show, our framework outperforms all comparative paradigms consistently in terms of the F1 score, while significantly reducing the Empty Rate.

The results reveal several key insights: *(1) Generic Retrieval Strategies Fail on Complex VKGs.* Standard RAG performs poorly on NPD (F1 $\approx$ 0.03), showing that generic text retrieval cannot handle large-scale schemas. Our framework, even without iteration (Table 3), substantially improves performance (F1 from 0.03 to 0.20) by integrating execution feedback. However, the task remains challenging, highlighting the need for further comprehensive techniques, such as the VKG enrichment proposals generated by our framework.

(2) Iterative Verification Bridges Executability and Correctness. A notable observation emerges from the results: on

[1] https://ontop-vkg.org/

Method	Usage		BGEE		NPD	
	Onto.	Map.	F1 ↑	Empty ↓	F1 ↑	Empty ↓
gpt-4o-mini						
vanilla LLM	✓		0.19	0.52	-	-
		✓	0.08	0.42	-	-
	✓	✓	0.23	0.51	-	-
RAG	✓		0.16	0.54	0.03	0.91
		✓	0.10	0.44	0.02	0.85
	✓	✓	0.21	0.41	0.03	0.81
ReAct	✓	✓	0.18	**0.34**	0.04	0.68
Ours	✓	✓	**0.30**	0.38	**0.13**	**0.58**
gpt-4o						
vanilla LLM	✓		0.27	0.43	-	-
		✓	0.16	0.40	-	-
	✓	✓	0.29	0.34	-	-
RAG	✓		0.19	0.51	0.03	0.96
		✓	0.20	0.41	0.03	0.82
	✓	✓	0.24	0.41	0.03	0.87
ReAct	✓	✓	0.20	0.37	0.03	0.71
Ours	✓	✓	**0.37**	**0.24**	**0.25**	**0.40**

Table 2: Comparative performance of query generation paradigms on BGEE and NPD. "-": context limit exceeded.

Variant	BGEE		NPD	
	F1 ↑	Empty ↓	F1 ↑	Empty ↓
Full Model	**0.37**	**0.24**	**0.25**	**0.40**
- w/o Ontology Retrieval	0.31	0.24	0.12	0.44
- w/o Mapping Retrieval	0.25	0.35	0.15	0.45
- w/o Memory Bank	0.28	0.32	0.20	0.53

Table 3: Ablation study on component contributions (*gpt-4o*).

BGEE, ReAct achieves a lower Empty Rate than our method (e.g., 0.34 vs. 0.38 with gpt-4o-mini) yet substantially lower F1 (0.18 vs. 0.30). This discrepancy suggests that ReAct often produces executable queries that fail to capture the information need. In contrast, our iterative mechanism retrieves additional context and leverages execution feedback to progressively refine queries, enabling more accurate capture of information needs and achieving higher F1.

(3) Scaling Model Capacity Cannot Replace Effective Framework Design. Although upgrading from *gpt-4o-mini* to *gpt-4o* yields consistent gains (e.g., our F1 on BGEE rises from 0.30 to 0.37), it does not close the performance gap: RAG and ReAct on NPD remain largely ineffective even with the stronger backbone model. This indicates that IN-VKGE requires task-specific designs tailored to VKG features, which general-purpose paradigms cannot provide.

Ablation Study

To understand the contribution of each VKG component, we conducted ablation experiments by removing ontology retrieval, mapping retrieval, or memory bank (as shown in Table 3): *(1) Ontology Retrieval* removal also leads to F1 drops of 0.06 on BGEE and 0.13 on NPD, indicating that semantic structure is essential for query generation. *(2) Mapping Retrieval* removal causes F1 drops of 0.12 on BGEE and 0.10 on NPD. This is because mappings ground ontology concepts in the database schema, and their absence leads to queries referencing unmapped concepts (hence the elevated Empty Rate).

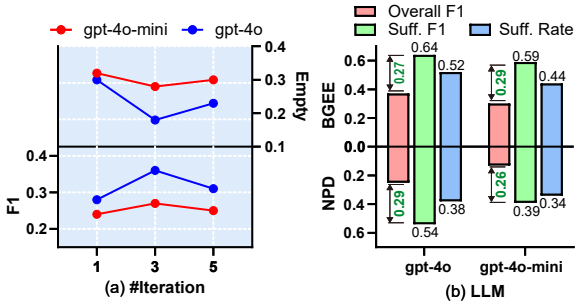

Figure 3: (a) Iteration count impact on BGEE. (b) F1 for all queries vs. a_i=SUFFICIENT queries.

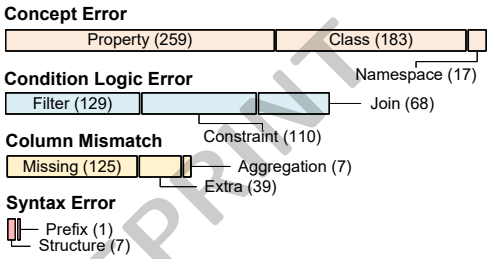

Figure 4: Distribution of error types in generated SPARQL queries.

(3) Removing the *Memory Bank* (i.e., single-iteration execution) reduces F1 by 0.09 on BGEE and 0.05 on NPD, indicating that iterative improvement leverages execution feedback to improve query quality.

Effect of Maximum Iterations

To understand how the iteration limit affects performance, we conducted experiments varying the maximum iterations (Iter $\in \{1, 3, 5\}$) on the BGEE dataset. From Figure 3(a), we observe that *Iter=3 achieves the optimal balance* between performance and exploration depth. Compared to single-iteration (Iter=1), three iterations improve the overall F1 from 0.24 to 0.27 (*gpt-4o-mini*) and from 0.28 to 0.36 (*gpt-4o*), while reducing the Empty Rate from 0.32 to 0.28 and from 0.30 to 0.18, respectively. However, excessive iterations (Iter=5) lead to performance degradation due to accumulated error propagation and context overload.

Fine-grained Error Analysis

To better understand the limitations of query generation, we conducted an error analysis on 331 incorrectly predicted samples (F1 < 1), collected from *gpt-4o* and *gpt-4o-mini* across both datasets. Note that a single query may exhibit multiple error types simultaneously. Figure 4 presents the hierarchical distribution of error categories. The errors are organized into four main categories: *(1) Concept Error*: involves incorrect usage of ontology URIs (properties, classes, or namespaces); *(2) Condition Logic Error*: mismatches in query logic (filters, joins, over/under-constrained conditions); *(3) Column Mismatch*: discrepancies in returned columns (missing, extra, or wrong aggregation); *(4) Syntax Error*: denotes SPARQL parsing failures. The error distribution shows that *Concept Errors* dominate, highlighting the challenge of

Information Need: List gene name, species scientific name, anatomical part name, and expression score for all records.

Exploration Trajectory:
- **Round 1** : Explore `Gene` and `Expression` properties separately. → **Result**: Both return rows. `Gene` has `rdfs:label`, `orth:organism`; Expression has `:hasExpressionLevel`, `:hasSequenceUnit`. Components work individually.
- **Round 2** : Try `Gene` → `Species` → `scientificName` chain. → **Result**: empty. `orth:organism` links to species ID, but `up:scientificName` not accessible.
- **Round 3** : Attempt 4-way join (`Gene` → `Species` → `Anatomy` → `Score`). → **Result**: Empty. The Gene-Species-ScientificName chain is broken.

Identified Gaps:
- Missing Mapping: No direct orth:Organism → scientificName property path exists.
- ...

Enrichment Proposals:
- Modify the ontology: Add up:scientificName to orth:Organism.
- Modify the mappings:

```
ID      Organism_ScientificName
Target  :organism/{speciesId} a orth:Organism ;
        up:scientificName {scientificName} .
Source  SELECT speciesId, CONCAT(genus, ' ', species) AS
        scientificName  FROM species
```

Figure 5: Case study on BGEE (MAXREACHED). (Green: execution results; Red: diagnostic observations.)

Information Need: List all fixed facility names and the water depth at their locations.

Exploration Trajectory:
- **Round 1** : Query using standard `rdfs:label`. → **Result**: Empty. Predicate mismatch
- **Round 2** : Where `{?s a npdv:FixedFacility. ?s ?p ?o}`. → **Result**: return rows. Discovered `npdv:name`.
- **Round 3** : Revised query using `npdv:name`. → **Result**: return rows. Correct property identified; FixedFacility retrieval successful.

Identified Gaps:
- Missing Standard Label: `npdv:FixedFacility` lacks `rdfs:label`. It relies solely on `npdv:name`, causing the initial discovery failure.
- ...

Enrichment Proposals:
- Modify the ontology: Map the facility name to `rdfs:label` (in addition to `npdv:name`) to support standard queries.
- Modify the mappings:

```
ID      FixedFacility_Name
Target  npd:facility/{fclNpdidFacility} rdfs:label
        {fclName}^^xsd:string .
Source  SELECT "fclNpdidFacility", "fclName" FROM
        "facility_fixed"
```

Figure 6: Case study on NPD (SUFFICIENT).

grounding natural language terms to ontology URIs in complex VKG schemas.

6.2 Evaluation of Enrichment Proposals

The primary goal of IN-VKGE is to assess whether an information need can be supported by the current VKG specification, and to provide actionable enrichment proposals. We evaluate this capability through decision reliability assessment, case studies, and expert evaluation.

Decision Reliability Assessment

When generating the final SPARQL query, our framework terminates with a decision indicating whether the information need can be supported. Figure 3(b) shows that performance is much better on SUFFICIENT queries (e.g., on BGEE, *gpt-4o*'s F1 rises from 0.37 Overall to 0.64 Sufficient), confirming the reliability of this self-assessment. This distinction helps guide the *Expert-in-the-Loop* process: *(1)* MAXREACHED serves as a warning for *missing information*, flagging needs that potentially require VKG enrichment; *(2)* SUFFICIENT indicates a working query was found, but may still suggest *better practices* to improve future usability.

Case Study

We present two representative reports (r) to illustrate how our framework identifies mapping gaps and generates enrichment proposals for expert review: *(1) BGEE Case (*MAXREACHED*):* As shown in Figure 5, the exploration reveals that `up:scientificName` is inaccessible via `orth:organism`, breaking the property chain. To address this gap, the framework proposes enriching the ontology vocabulary by adding `up:scientificName` to `orth:Organism` with new R2RML mappings. *(2) NPD Case (*SUFFICIENT*):* As shown in Figure 6, the initial query using `rdfs:label` returns empty due to predicate mismatch. Through iterative exploration, the framework identifies that the VKG uses a non-standard `npdv:name` property, which successfully fulfills the information need. Based on this finding, the enrichment proposal suggests mapping facility names to `rdfs:label` to support standard queries.

Dataset	Ident.	Validity	Exec.	Total
BGEE	1.78±0.06	1.76±0.11	1.62±0.23	5.16±0.39
NPD	1.78±0.14	1.75±0.19	1.58±0.30	5.11±0.62

Table 4: Expert evaluation of enrichment proposals.

Expert Evaluation of Enrichment Proposals

To assess the quality of enrichment proposals generated by *gpt-4o*, we conducted an expert evaluation study on all 198 samples from BGEE and NPD. Each sample was independently scored by three VKG experts on three dimensions (0-2 scale each): *Identification* (correctness of root cause identification), *Validity* (logical soundness of enrichment proposals), and *Executability* (readiness of generated code for direct use). The detailed evaluation guidelines, which were provided to the three experts, can be found in Appendix E. Table 4 presents the evaluation results across the three experts, showing that our framework achieves high scores on identification, validity, and executability.

7 Conclusion

In this work, we formulated IN-VKGE, the task of using information needs to guide VKG enrichment. We proposed an iterative framework that leverages LLMs to assess supportability, generate queries, and provide enrichment proposals. We also constructed the first manually annotated dataset for IN-VKGE. Evaluations demonstrate that the framework outperforms existing paradigms in query generation and produces high-quality enrichment proposals. By mitigating the coverage gaps between information needs and VKGs, this framework enables VKGs to evolve dynamically, reducing the reliance on manual intervention by experts. In the future, we aim to expand datasets to cover diverse domains and integrate the framework with existing VKG systems or tools.

Acknowledgments

This work is partially supported by National Nature Science Foundation of China under No. 62476058. We thank the Big Data Computing Center of Southeast University for providing the facility support on the numerical calculations in this paper. This research has been partially supported by the HEU project CyclOps (GA n. 101135513), by the Province of Bolzano and FWF through project OnTeGra (DOI 10.55776/PIN8884924) by the Province of Bolzano and EU through projects ERDF-FESR 1078 CRIMA, and ERDF-FESR 1047 AI-Lab, and by MUR through the PRIN project 2022XERWK9 S-PIC4CHU. This work has been carried out while Marco Di Panfilo was enrolled in the Italian National Doctorate on Artificial Intelligence run by Sapienza University of Rome in collaboration with Free University of Bozen-Bolzano. Linfang Ding is supported by the Start-up Funding from Hohai University (B250201131) and the Jiangsu Specially Appointed Professors Program.

Contribution Statement

*Corresponding author: guohui.xiao@seu.edu.cn.

References

[Allemang and Sequeda, 2024] Dean Allemang and Juan Sequeda. Increasing the accuracy of LLM question-answering systems with ontologies. *Proc. of ISWC'2024*, 2024.

[Anthropic, 2025] Anthropic. Introducing claude sonnet 4.5. *Anthropic blog*, 2025.

[Aumueller *et al.*, 2005] David Aumueller, Hong Hai Do, Sabine Massmann, et al. Schema and ontology matching with COMA++. In *Proc. of SIGMOD'2005*, pages 906–908, 2005.

[Avila *et al.*, 2024] Caio Viktor S. Avila, Vânia M. P. Vidal, Wellington Franco, et al. Experiments with text-to-SPARQL based on ChatGPT. In *Proc. of IEEE ICSC'2024*, pages 277–284, 2024.

[Azaria *et al.*, 2024] Amos Azaria, Rina Azoulay, and Shulamit Reches. ChatGPT is a remarkable tool—for experts. *Data Intelligence*, 6(1):240–296, 2024.

[Berant *et al.*, 2013] Jonathan Berant, Andrew Chou, Roy Frostig, et al. Semantic parsing on freebase from question-answer pairs. In *Proc. of EMNLP'2013*, pages 1533–1544, 2013.

[Bizer and Seaborne, 2004] Christian Bizer and Andy Seaborne. D2RQ-treating non-RDF databases as virtual RDF graphs. In *Proc. of ISWC'2004*, 2004.

[Brei *et al.*, 2025] Felix Brei, Lorenz Bühmann, Johannes Frey, et al. ARUQULA - an LLM based Text2SPARQL approach using ReAct and knowledge graph exploration utilities. In *Proc. of TEXT2SPARQL Challenge at ESWC'2025*, volume 4094 of *CEUR Workshop Proceedings*, pages 49–63, 2025.

[Calvanese *et al.*, 2007] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, et al. Ontology-based database access. In *SEBD*, pages 324–331, 2007.

[Calvanese *et al.*, 2017] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, et al. Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8(3):471–487, 2017.

[Calvanese *et al.*, 2021] Diego Calvanese, Davide Lanti, Tarcisio Mendes De Farias, et al. Accessing scientific data through knowledge graphs with ontop. *Patterns*, 2(10), 2021.

[Chen, 2024] Huajun Chen. Large knowledge model: Perspectives and challenges. *Data Intelligence*, 6(3):587–620, 2024.

[Cima *et al.*, 2023] Gianluca Cima, Antonella Poggi, and Maurizio Lenzerini. The notion of abstraction in ontology-based data management. *Artificial Intelligence*, 323:103976, 2023.

[Daraio *et al.*, 2016] Cinzia Daraio, Maurizio Lenzerini, Claudio Leporelli, et al. Data integration for research and innovation policy: An ontology-based data management approach. *Scientometrics*, 106(2):857–871, 2016.

[De Giacomo *et al.*, 2012] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, et al. MASTRO: A reasoner for effective ontology-based data access. In *Proc. of ORE'2012*, volume 858, 2012.

[de Medeiros *et al.*, 2015] Luciano Frontino de Medeiros, Freddy Priyatna, and Óscar Corcho. MIRROR: Automatic R2RML mapping generation from relational databases. In *Proc. of ICWE'2015*, volume 9114, pages 326–343, 2015.

[DeepSeek-AI, 2024] DeepSeek-AI. Deepseek-V3 technical report, 2024.

[Dubey *et al.*, 2019] Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, et al. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *Proc. of ISWC'2019*, pages 69–78. Springer, 2019.

[Emonet *et al.*, 2024] Vincent Emonet, Jerven Bolleman, Séverine Duvaud, et al. LLM-based SPARQL query generation from natural language over federated knowledge graphs. *Proc. of ISWC'2024 Special Session on Harmonising Generative AI and Semantic Web Technologies*, 2024.

[Gao *et al.*, 2023] Yunfan Gao, Yun Xiong, Xinyu Gao, et al. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[Google, 2025] Google. Gemini 3 pro: Best for complex tasks and bringing creative concepts to life. *Google blog*, 2025.

[Herzig and Berant, 2017] Jonathan Herzig and Jonathan Berant. Neural semantic parsing over multiple knowledge-bases. In *Proc. of ACL'2017*, pages 623–628, 2017.

[Hovland *et al.*, 2017] Dag Hovland, Roman Kontchakov, Martin G. Skjæveland, et al. Ontology-based data access to Slegge. In *Proc. of ISWC'2017*, pages 120–129. Springer, 2017.

[Jiang *et al.*, 2023] Jinhao Jiang, Kun Zhou, Zican Dong, et al. StructGPT: A general framework for large language model to reason over structured data. In *Proc. of EMNLP'2023*, pages 9237–9251, 2023.

[Jiménez-Ruiz *et al.*, 2015] Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, et al. BootOX: Practical mapping of RDBs to OWL 2. In *Proc. of ISWC'2015*, pages 113–132, 2015.

[Kharlamov *et al.*, 2015] Evgeny Kharlamov, Dag Hovland, Ernesto Jiménez-Ruiz, et al. Ontology based access to exploration data at statoil. In *Proc. of ISWC'2015*, pages 93–112. Springer, 2015.

[Kuhn, 1955] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[Lanti *et al.*, 2015] Davide Lanti, Martin Rezk, Guohui Xiao, et al. The NPD benchmark: Reality check for OBDA systems. In *Proc. of EDBT'2015*, pages 617–628, 2015.

[Lenzerini, 2011] Maurizio Lenzerini. Ontology-based data management. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 5–6, 2011.

[Luo *et al.*, 2024] Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, et al. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *Proc. of ICLR'2024*, 2024.

[Ma *et al.*, 2025a] Chuangtao Ma, Yongrui Chen, Tianxing Wu, et al. Large language models meet knowledge graphs for question answering: Synthesis and opportunities. In *Proc. of EMNLP'2025*, pages 24589–24608, 2025.

[Ma *et al.*, 2025b] Shengjie Ma, Chengjin Xu, Xuhui Jiang, et al. Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation. In *Proc. of ICLR'2025*, 2025.

[OpenAI, 2025] OpenAI. Introducing gpt-5.2. *OpenAI blog*, 2025.

[Pinkel *et al.*, 2013] Christoph Pinkel, Carsten Binnig, Evgeny Kharlamov, et al. IncMap: pay as you go matching of relational schemata to OWL ontologies. In *OM*, pages 37–48, 2013.

[Rashid and McGuinness, 2025] Sabbir Rashid and Deborah L. McGuinness. Designing and evaluating an ensemble reasoning-based clinical decision support system. *Data Intelligence*, 7(1):1–39, 2025.

[Rodriguez-Muro *et al.*, 2013] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontology-based data access: Ontop of databases. In *Proc. of ISWC'2013*, pages 558–573. Springer, 2013.

[Sequeda and Miranker, 2017] Juan F. Sequeda and Daniel P. Miranker. A pay-as-you-go methodology for ontology-based data access. *IEEE Internet Computing*, 21(2):92–96, 2017.

[Sequeda *et al.*, 2023] Juan Sequeda, Dean Allemang, and Bryon Jacob. A benchmark to understand the role of knowledge graphs on large language model's accuracy for question answering on enterprise sql databases, 2023.

[Soru *et al.*, 2017] Tommaso Soru, Edgard Marx, Diego Moussallem, et al. SPARQL as a foreign language. In *SEMANTiCS (Posters & Demos)*, 2017.

[Soylu *et al.*, 2014] Ahmet Soylu, Evgeny Kharlamov, Dmitriy Zheleznyakov, et al. OptiqueVQS: Visual query formulation for OBDA. CEUR Workshop Proceedings, 2014.

[Sun *et al.*, 2024] Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, et al. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *Proc. of ICLR'2024*, 2024.

[Wang *et al.*, 2024] Yu Wang, Nedim Lipka, Ryan A. Rossi, et al. Knowledge graph prompting for multi-document question answering. In *Proc. of AAAI'2024*, pages 19206–19214, 2024.

[Xiao *et al.*, 2018] Guohui Xiao, Diego Calvanese, Roman Kontchakov, et al. Ontology-based data access: A survey. 2018.

[Xiao *et al.*, 2019] Guohui Xiao, Linfang Ding, Benjamin Cogrel, et al. Virtual knowledge graphs: An overview of systems and use cases. *Data Intelligence*, 1(3):201–223, 2019.

[Xiao *et al.*, 2025] Guohui Xiao, Lin Ren, Guilin Qi, et al. LLM4VKG: Leveraging large language models for virtual knowledge graph construction. In *Proc. of IJCAI'2025*, pages 4715–4723, 2025.

[Yao *et al.*, 2023] Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. React: Synergizing reasoning and acting in language models. In *Proc. of ICLR'2023*, 2023.

[Yu *et al.*, 2018] Tao Yu, Rui Zhang, Kai Yang, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proc. of EMNLP'2018*, 2018.

[Zahera *et al.*, 2024] Hamada M. Zahera, Manzoor Ali, Mohamed Ahmed Sherif, et al. Generating SPARQL from natural language using chain-of-thoughts prompting. In *Proc. of SEMANTiCS'2024*, pages 353–368, 2024.

[Zhang *et al.*, 2024] Zhiqiang Zhang, Liqiang Wen, and Wen Zhao. A GAIL fine-tuned LLM enhanced framework for low-resource knowledge graph question answering. In *Proc. of CIKM'2024*, pages 3300–3309, 2024.

[Zhang, 2023] Jiawei Zhang. Graph-toolformer: To empower LLMs with graph reasoning ability via prompt augmented by ChatGPT. *arXiv preprint arXiv:2304.11116*, 2023.