Evaluating Prompting Methodologies for Generating Federated SPARQL Queries over Biomedical Knowledge Graphs

Sebastian Schmülling
Department of Advanced Computing Sciences
Faculty of Science and Engineering
Maastricht University
Maastricht, The Netherlands

***Abstract*—SPARQL is a standardized query language that can be used to retrieve information from distributed RDF databases called SPARQL endpoints. Formulating sophisticated SPARQL queries over RDF KGs requires substantive expertise in SPARQL and familiarity with the KG schema or schemas in the case of queries across multiple SPARQL endpoints. Large Language Models (LLMs) have emerged as surprisingly effective tools for knowledge translation and several attempts have leveraged LLMs to generate SPARQL queries from user questions. This work reports on an LLM-based approach to construct federated SPARQL queries over multiple RDF KGs. This work evaluates methodologies such as *chain-of-thought* and *least-to-most* prompting to generate federated SPARQL generation, focusing on a collection of biomedical KGs, and evaluating against expert pairs of experts created natural language questions and federated SPARQL queries. The results show that context in the form of relevant query examples, combined with prompt engineering, retrieval augmented generation, and syntactic query validation, significantly enhances the generation of federated SPARQL queries.**

***Index Terms*—SPARQL Generation, LLM, Natural Language Query Generation, Prompt Engineering, Biomedical Knowledge Graphs**

I. Introduction

SPARQL is a standardized query language to retrieve information from data represented with the Resource Description Framework (RDF). Formulating expressive SPARQL queries over one or more RDF datasets or RDF Knowledge Graphs (KGs) is a time-consuming, iterative, and challenging task as it requires substantive expertise in the SPARQL language and familiarity with the schema(s) that underlie the KG. Consequently, there is ongoing interest in developing methods to automatically transform the inputs of domain experts into structured SPARQL queries for information retrieval. These inputs can then be made accessible to users with less expertise in SPARQL language and schema(s).

Knowledge Graph Question Answering (KGQA) systems aim to enable intuitive and accessible querying of KGs using natural language. Several approaches have been proposed for SPARQL query generation ranging from filling in templates [1] [2] [3] to developing novel sequence-to-sequence generative models [4], and using retrieval-augmented generation (RAG) [5]. Other approaches use linguistic parsing and heuristic rules [6]. While promising, RAG systems face significant challenges in generating accurate SPARQL queries, particularly for arbitrary, large-scale, and evolving KGs. The difficulty increases for the composition of federated queries, which requires semantic equivalences to be identified across different KGs. This added complexity of federated queries increases the likelihood of errors and makes SPARQL query generation even more challenging.

While prompting strategies could facilitate generating SPARQL queries, their effectiveness for more complex federated queries remains uncertain. Prompting strategies such as chain-of-thought (CoT) reasoning offer partial improvements but can fall short in generalizing to longer and more intricate queries. CoT struggles to generalize beyond the complexity of examples provided in its prompt [7]. This limitation is particularly relevant to SPARQL query generation, especially in federated environments, where query complexity can vary widely and often surpass the difficulty of the examples provided during prompting. To address these generalization issues, the least-to-most (LtM) prompting strategy, offers a more robust approach [7]. LtM decomposes complex problems into simpler sub-problems, solved sequentially, where each solution builds on previous answers. Typically, this two-stage approach uses few-shot prompting: first decomposing, then solving each subproblem in sequence. Unlike CoT prompting which attempts to solve the problem holistically, LtM's structured sequential approach demonstrates superior generalization capabilities for complex problems [7]. However, to current knowledge, this methodology has not yet been evaluated for SPARQL query generation.

This leads to the following *problem statement*: The central problem this thesis addresses is the complex task of generating syntactically correct and semantically accurate federated SPARQL queries across multiple, heterogeneous biomedical KGs. LLMs offer considerable potential to overcome these challenges by generating SPARQL queries from natural

This thesis was prepared in partial fulfilment of the requirements for the Degree of Bachelor of Science in Data Science and Artificial Intelligence, Maastricht University. Supervisors: [Michel Dumontier and Chang Sun]

language. Still, their capabilities, particularly when prompting strategies like chain-of-thought (CoT) and Least-to-Most (LtM) are employed, in handling the intricacies of federated queries are not yet well understood. This represents a critical gap in advancing KGQA systems, especially for complex domains like biomedicine that heavily rely on integrated data from distributed KGs.

Key research challenges include achieving accurate ontology and schema grounding, especially the correct entity and class entity identifiers (URIs) grounding across different KGs to enable semantically correct federated SPARQL query generation. Establishing a comprehensive evaluation framework is vital, yet difficult due to inherent complexities of federated SPARQL queries.

This study investigated different prompting strategies, including CoT and LtM, to develop a KGQA system capable of handling federated queries across biomedical KGs. Central to the evaluated methodologies is a RAG component, applied consistently across strategies. It injects relevant entity and class URIs from a vector database into queries, thereby mitigating common URI hallucination problems. Specifically for LtM, a novel iterative validation phase for generated triple patterns was incorporated to confirm pattern viability prior to final query assembly. These approaches were evaluated on a curated subset of 32 complex federated queries derived from a collection of user posed questions and federated SPARQL pairs in the biomedical domain (2024) [8]. A multi-faceted evaluation framework was developed, incorporating an innovative approach to address challenges to evaluate the correctness of returned results of predicted queries.

The evaluation revealed substantial performance variations among prompting strategies. In particular, LtM demonstrated the highest performance with 96.6% of syntactic valid generated SPARQL queries, the best retrieval of meaningful, non-empty results, and leading performance with an F1 score of 0.51. LtM appeared as most robust and reliable generating these results, while accompanied by highest token usage among the evaluated prompting strategies. However, significant variability in LtM's token consumption, combined with its primary reliance on prompt tokens over typically more expensive completion tokens, leading to practical cost allocations comparable to CoT. These findings highlight the effectiveness of LtM's novel decompositional structure and its iterative validation process in navigating the complexities of federated SPARQL query generation.

To guide these investigations, experiments were conducted to answer the following research questions:

1) **How effective is the Chain-of-Thought (CoT) prompting methodology in generating accurate federated SPARQL queries across multiple bioinformatics knowledge graphs?**
2) **Can the Least-to-Most prompting methodology outperform the Chain-of-Thought approach in generating accurate federated SPARQL queries?**
3) **Does the potential improvement in accuracy provided by the Least-to-Most prompting methodology justify its higher computational cost compared to the Chain-of-Thought approach in federated SPARQL query generation?**

This paper is structured as follows. Section 2 reviews related work focusing on RAG-based KGQA systems and prompting methodologies. Section 3 describes the dataset used for our experiments and details the different prompting methodologies evaluated. Section 4 outlines the experimental setup and the evaluation framework employed to assess federated SPARQL query generation performance. Section 5 presents the results. Section 6 discusses the implications of these findings and limitations. Finally, Section 7 concludes the paper by answering the research questions and suggests potential directions for future research.

II. Related work

KGQA systems have been an active area of research for over a decade [9]. The emergence of LLMs has significantly advanced natural language queries, including their application to KG query generation [9].

A general key bottleneck is limited training data for converting natural-language questions into syntactically valid and semantically aligned SPARQL queries. This issue becomes even more significant for domain-specific knowledge graphs and in federated settings. To address these challenges a potential approach that has been presented involves fine-tuning an OpenLlama LLM for SPARQL query generation over life science KGs [9]. Fine-tuning the LLM on a dataset of question-SPARQL pairs from a domain-specific KG significantly improved the performance of generating SPARQL queries. Despite the increased performance, its generalizability to other domains may be limited, which is essential for generating federated SPARQL queries. Furthermore, it remains uncertain how this approach can accommodate new KGs or the evolving nature of existing ones.

Recent studies have begun to frame KGQA systems as a Retrieval-Augmented Generation (RAG) task. A representative example maintains a vector index of 360 manually crafted SPARQL templates [10]. For a given question the system retrieves templates with the highest semantic similarity scores. Subsequently it injects these templates, along with the KGs ontology and the user question, into an LLM that rewrites the best match into a complete SPARQL query. Providing the LLM with retrieved templates, rather than supplying only the user's question and ontology, yielded substantially higher performance [10]. While being promising, this approach may become infeasible for larger KGs. Injecting a KG's entire ontology, or those of several distinct KGs for federated queries, could quickly exhaust the LLM's context window size.

Yet another approach focuses on augmenting the LLM with an entire KG. In conventional RAG approaches, documents are split into small embedding-indexed chunks and retrieved by similarity. Applying this approach to semantic artifacts such as RDF or TTL files would destroy structural dependencies and erode semantic clues essential for precise query generation [11]. To address these challenges, the approach first identifies

entities in the user's question and subsequently maps them to their corresponding URIs from the KG. In the next step, a subgraph is constructed by using graph modularization techniques, which retrieve enough information to construct a SPARQL query. After subgraph construction, the subgraph, matched entities, and the users question are injected in a predefined prompt and send to the LLM. Missing entity links are recognized as the most common cause for errors in SPARQL query generation [11]. The applicability of this single-KG focused approach to constructing federated SPARQL queries is not straightforward and remains an open question.

Emonet et al. [5] proposed an LLM-based KGQA system tailored to biomedical KGs maintained by the Swiss Institute of Bioinformatics. LLMs and endpoint metadata are leveraged as a RAG system to dynamically generate SPARQL queries. The meta data comprises example queries from several KGs and human-readable Shape Expressions (ShEx) for each class. This work distinguishes itself by supporting metadata from multiple KGs, thereby representing an early effort to facilitate federated SPARQL query generation. The system does well on a limited set of evaluative pairs, but hallucinations of entity URIs highlight ongoing challenges, in this work and that of other's queries [12].

To address the challenges of generating accurate SPARQL queries on complex and federated KGs using natural language, prompt engineering has emerged as a complementary strategy. Recent methods such as Chain-of-Thought (CoT) and Least-to-Most (LtM) prompting have demonstrated improved reasoning and generalization in LLMs [13] [7]. Zahera et al. [14] apply CoT prompting to guide LLMs through structured reasoning steps for SPARQL generation, outperforming state-of-the-art methods on the benchmarks datasets QALD-10 and QALD-9 with an F1 score of 4.4% and 3.0% respectively. However, the approach was not evaluated on federated queries and uses only a basic CoT template, without tailoring the strategy to handle query decomposition across multiple KGs [15] [13].

Finally, a commonly recognized issue in KGQA systems is lack of standard benchmarks and comprehensive evaluations. Existing benchmarks for federated SPARQL querying such as the the billion-triple benchmark are often synthetic, small-scale, or lack fine-grained evaluation criteria [16]. The LLM-KG-Bench framework [12] offers a more extensible and automated evaluation setup, assessing syntax, semantic reading, and creation. However, its interactive correction loop better suits LLMs directly, rather than fully-automated KGQA systems that lack such feedback mechanisms.

As surveyed, existing RAG-based KGQA approaches struggle significantly with the demands of federated SPARQL query generation. Recognized challenges comprise generalizability, managing the combined context and schemas of multiple KGs, URI hallucinations - a general difficulty in SPARQL query generation, which becomes even more critical in federated queries (multiple schemas) - and the need for tailored prompting and reasoning strategies.

III. Dataset and Methodology

A. Methodology

As highlighted in related work, LLM-based KGQA systems face a key challenge when generating federated SPARQL queries - hallucinations of entity and endpoint URIs - which limits schema awareness, and robust reasoning over distributed KGs [5] [13]. To mitigate this, the methodology described incorporates a RAG component that retrieves valid URIs for entities and classes mentioned in the natural language question. This is used consistently across all prompting strategies. Specifically, an LLM is first used to identify key terms in the input question, then retrieves top-10 matching URIs per term from a vector store.

As can be seen in Figure 1 the overall system architecture incorporates this RAG component as an initial step. From a vector database, where relevant entity URIs and example question-SPARQL query pairs are indexed upon initial deployment, these retrieved entities are injected as context (Context A) for every prompting methodology (Figure 1). For certain methodologies - CP-A and LtM - the LLM's input is enriched by context B, consisting of example question-SPARQL query pairs relevant to the user's question. Based on these contextual inputs and methodology, the LLM generates a candidate SPARQL query.

The source code of the methodologies described herein, including all prompting templates and evaluation framework, as well as the dataset are available from Github.[1] We compare four prompting strategies for federated SPARQL generation over biomedical KGs: 1) baseline, 2) *construction prompt (CP)*, 3) *augmented construction prompt (CP-A)*, 4) *chain of thought (CoT)* prompting, and 5) *least to most (LtM)* prompting.

The **baseline** employs a basic prompt including the retrieved candidate entity identifiers, a single-sentence task description, and instructions specifying the desired output format.

The **CP** uses a prompt that provides explicit instructions to guide the LLM in creating a federated SPARQL query step by step, such as the requirement to include a `SERVICE` clause to access federated endpoints. The CP also includes the URIs and short descriptions of the relevant SPARQL endpoints against which the queries can be executed or federated. It helps the model to use the right endpoints and avoid hallucinations of endpoint URIs.

The **CP-A** further augments CP with few-shot question-query examples dynamically retrieved from a vector store. The few-shot examples are collected from some public KGs which provide sample queries to guide new users in exploring the databases [9]. The CP-A provides the LLM with a richer contextual understanding of endpoint schemas and ontologies by augmenting the standard CP with ten relevant example question-SPARQL query pairs. Analogous to our entity retrieval process, these example pairs are embedded in a vector database, allowing dynamic retrieval of the most

[1] https://github.com/sebastians1999/federated-sparql-rag-agent.git

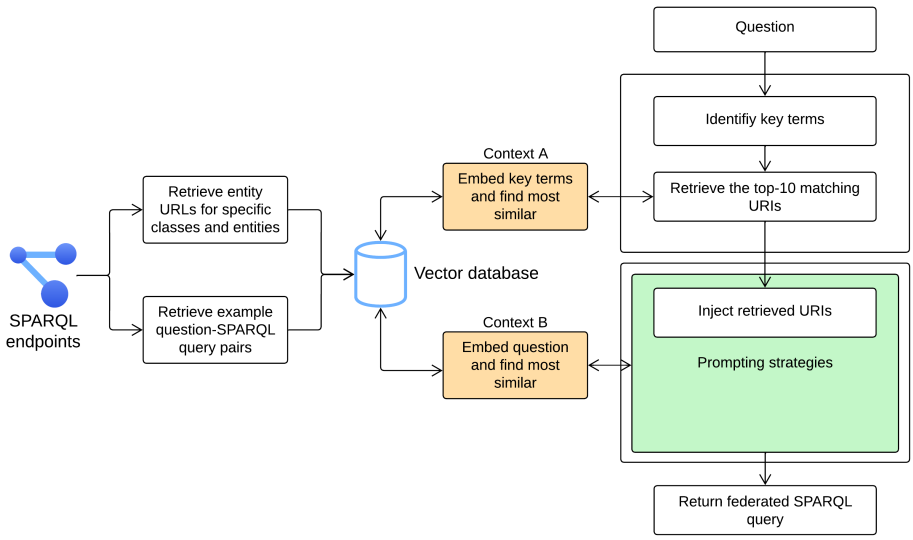


Fig. 1. **System architecture for federated SPARQL query generation. Context A contains valid URIs for classes and entities. Context B contains relevant example question-SPARQL query pairs and is only used by CP-A and LtM.**

relevant examples for a given user's question. The sensitivity to examples is key as varying the permutation of few-shot exemplars can highly influence the model's performance [13]. Studies show it can cause the accuracy of GPT-3 on SST-2 range from 54,3% to near state of the art with 93,4% [17]. Notably, while the majority of these illustrative queries are not inherently federated, they are selected for their contextual relevance. Consequently, they are expected to provide valuable insight into the structure and usage of individual endpoints, thereby assisting the LLM in constructing more accurate federated SPARQL queries.

The **CoT** strategy has two versions of prompts: 1) a vanilla CoT that simply adds "think step by step" to the CP prompt, and 2) a few-shot CoT prompting using three comprehensive examples. Each example presents a sophisticated, hand-curated 12-step reasoning process, from entity recognition to query assembly, to guide the LLM to generate valid federated SPARQL queries. Each example includes reasoning annotations to demonstrate intermediate logic and enhance generalization.

The 12-step reasoning process is as follows:

1) Identify main entities/classes.
2) Determine return labels (future SELECT variables).
3) Map entities to ontology IRIs (with the instruction: "If there is not a good match in the provided context, figure out a mapping from your knowledge").
4) Locate relevant SPARQL endpoints and determine the information to retrieve from each.
5) Choose the primary endpoint for query execution.
6) Identify join variables between endpoints.
7) Draft triple patterns for each endpoint.
8) Mark federated patterns with SERVICE clauses.
9) Assemble the complete WHERE block.
10) Incorporate any necessary filtering conditions or additional clauses (e.g., OPTIONAL, ORDER BY).
11) Write the SELECT clause using the variables determined in Step 2.
12) Define necessary namespace prefixes and specify the target endpoint if not already fully addressed by SERVICE clauses.

The **LtM** prompting strategy decomposes the generation task into three externally defined sub-problems: 1) query planning, 2) triple pattern generation and validation, and 3) final query assembly. The output of each subproblem is subsequently passed as input to the next, incrementally building the solution to the original problem.

1) In the planning phase, the LLM identifies relevant endpoints, maps entities to URIs, and drafts candidate SELECT variables using the same contextual inputs as CP-A.
2) In the triple generation and validation phase, the LLM constructs candidate triple patterns per endpoint, each wrapped in an ASK WHERE clause for validation. Each generated triple pattern undergoes a two-step validation process. First, it is checked for syntactic correctness. If valid, the corresponding `ASK WHERE` query is executed against the designated SPARQL endpoint. A return value of `True` indicates a match in the data; `False` or execution errors (e.g., due to schema mismatches) are treated as failures. In such cases, feedback is returned to the LLM, which is allowed up to three attempts to revise the pattern accordingly.
   Prompting is tailored to whether the target is a primary or federated endpoint. For the primary endpoint, the LLM receives the endpoint identifier, a description of the endpoint, URI mappings, relevant example queries,

and any previously failed patterns with associated error messages. For federated endpoints, the LLM is additionally provided with information about the primary endpoint to ensure coherent query composition across sources. All other contextual inputs remain the same. The LLM is explicitly instructed to wrap federated patterns within a `SERVICE` clause, forming part of the overall `ASK WHERE` query.

3) In the assembly phase, validated triple patterns are joined using shared variables to form the final federated SPARQL query. The LLM refines the `SELECT` clause and prefixes as needed.

B. Dataset

This work used a dataset [8] of human-written natural language questions paired with their corresponding SPARQL queries, including federated queries, over biomedical KGs. At time of download, the dataset comprised 1154 SPARQL queries and 96 federated SPARQL queries involving 12 SPARQL endpoints. To ensure entity URI indexing remained feasible, the analysis was focused on the top 3 co-occurring SPARQL endpoints. Since queries primarily federate among these endpoints, this provided valuable entity URIs for most of the cases. The set of federated SPARQL queries comprised: (i) **Uniprot**, a database of protein sequence and function information; (ii) **Rhea**, a database containing biochemical reactions, and (iii) **SwissLipids** a curated lipidomics knowledge base. After filtering, the resulting test set comprises 44 federated SPARQL queries. Of these, 9 returned errors, due to schema mismatches, and 3 produced no results during execution. Consequently, a final evaluation set of 32 valid federated SPARQL queries.

To characterize the degree of federation for the three endpoints in the evaluation set, SwissLipids, Uniprot, and Rhea federated with an average of 1.75, 1.30, and 1.00 with other endpoints, respectively. To mitigate potential bias from duplicates among the few-shot CoT examples - one chosen per endpoint - the SPARQL query similarity within the evaluation set was calculated using Levenshtein similarity score, a validated metric for this purpose [18]. Similarities in Figure 2 show that there are no duplicates. Notably for the SwissLipids few-shot example, two queries show a higher similarity of 0.75 and 0.74. Moreover, an exhaustive comparison of all queries in the evaluation set against each other yielded no duplicates with a highest similarity of 0.92 (Appendix B).

IV. Experiments and Evaluation

A. Experiment setting

All experiments were conducted on a MacBook with an Apple M2 chip, 8GB RAM, and a 500GB SSD. LLM workflows used the Langchain framework, with all methodologies evaluated using the Google Gemini-2.0-Flash-001 LLM. The LLM was chosen based on free-tier rate limitations. Exact specifications of the model are undisclosed from the provider.

To extract key terms for the RAG component, a temperature of 0.2 and top-p = 1.0 was used. All other generation steps

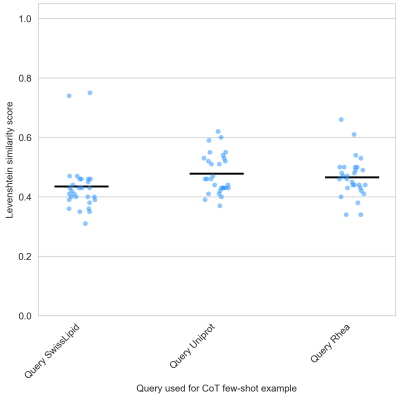


Fig. 2. **Levenshtein similarity score of evaluation set SPARQL queries to few-shot examples.**

(including query construction and prompting) were run with temperature = 0.1 and top-p = 1.0. These parameters influence the LLM's output: temperature (normally set between 0.0 and 1.0) influences randomness, with lower values making the output more deterministic and reducing variability by favoring highly probable tokens, while top-p (set to 1.0) allows the model to select from all tokens based on their temperature-scaled probability.

The chosen temperature values reflect findings from experimental runs of the evaluation set across different temperatures and methodologies. These initial evaluations indicated a general trend of improved performance for lower temperatures. For instance, reducing the CoT temperature from 0.8 to 0.1 increased the rate of syntactically valid SPARQL queries from 50% to 71.9%. Relying on low temperatures for tasks requiring precision and factual accuracy aligns with prior research [19].

Qdrant served as the vector store. Dense, sparse, and hybrid retrieval strategies were evaluated on a 620,000 item subset comprising ChEBI chemicals, UniProt diseases, and approximatly 600,000 UniProt taxons. For dense embeddings, only the small embedding model - BAAI/bge-small-en-v1.5 - was considered, due to hardware limitations for larger models. Performance was measured on 3,798 randomly sampled entities, with their labels as queries, using a top-10 cutoff. Results were compared via a two-sided paired Student's t-test (p-value = 0.05). While dense and hybrid retrieval showed the best performance, hybrid retrieval only performed slightly better across metrics (Appendix C). Observations showed computing the sparse embeddings takes 7-8 times longer than the dense embeddings per batch of size 144. Given evaluation results and faster embedding time, dense retrieval was chosen.

The data volume - UniProt (150 billion triples), Rhea (5 million triples), and SwissLipids (15 million triples) - made comprehensive indexing of all entities and classes unfeasible. After carefully analyzing the schemas and federated SPARQL queries, it was decided to index the following entities:

- **(i) UniProt:** Diseases and taxons.
- **(ii) Rhea:** Rhea reactions and chemical compounds from ChEBI.

- **(iii) SwissLipids:** No entities from this knowledge graph were directly indexed. This decision was based on the observation that, for the types of entities and relationships relevant to the evaluation queries, SwissLipids primarily utilizes or links to entities within the UniProt and Rhea namespaces.

The collection comprised approximately 2,651,000 indexed entities and classes.

For retrieving relevant few-shot question-query examples, the large model - BAAI/bge-large-en-v1.5 - was chosen. This choice was driven by its superior ability to achieve a more nuanced understanding of SPARQL queries, capturing deeper semantic properties that smaller models might miss. In both RAG components cosine similarity was employed for nearest-neighbor retrieval.

For the execution of predicted federated SPARQL queries against the respective endpoints, a uniform timeout limit of 10 minutes per query was enforced. The variability in outputs generated by LLMs can lead to performance fluctuations, even under consistent experimental settings. To account for this and to enhance the reliability of findings, each methodology was evaluated by conducting 10 independent generation trials for every query.

B. Evaluation methods

All proposed methodologies were evaluated using three complementary evaluation strategies: execution-based, structural, and content-based metrics to measure the quality of generated SPARQL queries. An overview of the evaluation framework can be seen in Figure 3. After a SPARQL query has been generated, a Execution-Based Evaluation assesses syntactic validity. Is this given the queries are executed against the predicted target endpoint and results are categorized. For successfully executed queries, Content-Based Evaluation assesses semantic accuracy: output columns from predicted and ground truth queries are aligned. Then, the precision, recall, and F1 score of these matched result columns are calculated. Finally, for all predicted queries Query Structure Similarity Evaluation - SP-BLEU and METEOR - is used to compare against ground truth query structures (Figure 3).

Execution-Based Evaluation. The first step evaluates the syntactic correctness and executability of each generated SPARQL query. Syntactic correctness is verified using an open-access SPARQL validator[2]. Queries failing this check are excluded from further evaluation. Valid queries are then executed against their predicted target endpoints. Execution results are categorized as: (i) successfully executes and returns data; (ii) successfully executes but yields an empty result set; (iii) encounters an error during execution. In this study, these execution errors fell into two categories: a semantic mismatch between the generated query and the schema at the target endpoint or federated endpoint(s), or an incorrect URL for one of the specified SPARQL endpoints; or (iv) results in a timeout (more than 10 minutes).

[2]http://www.sparql.org/query-validator.html

Query Structure Similarity Evaluation. To assess the structural similarity between generated and reference SPARQL queries [9], two metrics that were designed for evaluating the output of machine translation systems are applied: SP-BLEU [20] [9] and METEOR [21] [9]. **SP-BLEU** is a modified version of BLEU, specifically designed for SPARQL query. It preserves query structure while ignoring variable naming differences. **METEOR** improves over traditional metrics like BLEU by incorporating linguistic features such as synonymy, stemming, and word order, and placing more emphasis on recall to better align with human judgments of translation quality.

Content-Based Evaluation. Structurally different SPARQL queries may still yield equivalent results. To evaluate the correctness of returned answers, the output tables produced by the generated with those of corresponding reference queries were compared. A key challenge in this comparison is the potential mismatch in column labels, as differences in variable naming hinder direct comparison. To address this, columns are aligned by first attempting exact matches through case-insensitive and whitespace-trimmed comparisons.

For the remaining unmatched labels, a semantic similarity approach is applied: Initial evaluations of string matching techniques, specifically Levenshtein distance and n-gram matching, on observed examples demonstrated suboptimal performance. Notably despite semantic equivalence (e.g., between ”organism” and ”species”), the resulting similarity scores were considerably low. Addressing this limitation the focus was shifted towards the use of text embeddings. Three embedding models were evaluated: BAAI/bge-large-en-v1.5, BAAI/bge-base-en-v1.5 and BAAI/bge-small-en-v1.5 with dimensions of 384, 768 and 1024 respectively. These were selected in consideration of hardware limitations, owing to their computationally efficient nature and fast processing capabilities. The test set comprised 10 observed examples from experimental runs (Appendix A). Table I shows the medium and large embedding model achieved highest accuracy, with the larger model better capturing label nuances and achieving higher mean similarity to ground truth labels, leading to its selection.

After column labels are embedded, pairs are established when they are mutual best matches between generated and reference label, with cosine similarity above 0.7 and each predicted label matched only once. The threshold was found to be good after inspecting the test set manually. Once all possible column pairs are identified, the aligned columns are converted into sets of row-level tuples and compute precision,

TABLE I
EMBEDDING MODEL PERFORMANCE COMPARISON

Model	Accuracy	Mean True Pair Similarity
BAAI/bge-large-en-v1.5	0.90	0.77
BAAI/bge-small-en-v1.5	0.80	0.76
BAAI/bge-base-en-v1.5	0.90	0.75

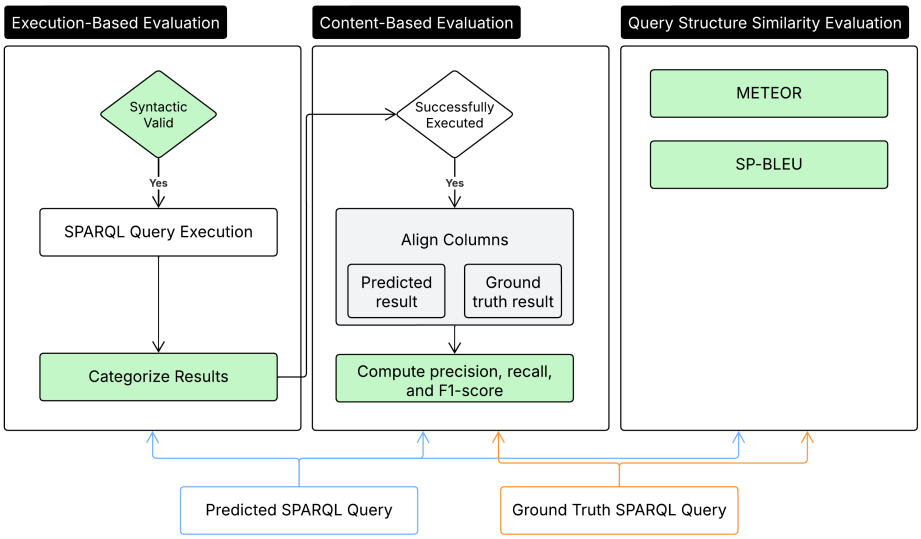


Fig. 3. **Overview of Evaluation Framework.**

recall, and F1-score to assess the semantic accuracy of the generated query results.

While all evaluation strategies provide valuable information, the Content-Based Evaluation is the most crucial for assessing the correctness of the generated queries. Execution-based Evaluation serves as an essential preliminary check for syntactic validity and executability. Although indispensable, its usefulness is limited to these initial aspects. It offers little information about the semantic accuracy or relevance of the query's output, both of which are of primary importance. For federated SPARQL queries in particular, structural similarity is not that decisive because the query can be structured and linked in many different ways, while still retrieving the correct results. Nevertheless, this form of evaluation can provide useful information, such as confirming the presence of expected core triple patterns, or consistently used namespace prefixes. These reflect the query intent despite structural variations. However, Content-Based Evaluation relies on approximation, which limits its reliability.

V. Experiment Results

Figure 4 illustrates considerable variations in syntactic validity of generated SPARQL queries across methodologies, with LtM achieving the highest performance of 96.6%. This significantly surpassed all other methodologies, including the Baseline, which only achieved a syntactic mean validity of 6.9%. Introducing structured prompting via the Construction Prompt (CP) increased mean validity to 62.2%, and further incorporating few-shot question-query examples (CP-A) raised it to 89.1%, reflecting a 27.1% improvement over CP alone.

After satisfying the requirement of syntactic correctness, the execution behavior of the queries at the endpoints was examined (Figure 5). The Baseline consistently failed at this stage, with all syntactically valid queries resulting in execution errors. CP, CoT, and few-shot CoT showed the lowest error rate, with few-shot CoT achieving the best result at 6.7%. However, these methods also exhibited relatively high rates of empty results: 68.3% for CP, 73.7% for CoT, and 57.4% for few-shot CoT. Despite having a slightly higher error rate, CP outperformed CoT in producing successful results, returning non-empty outputs in 8.0% of cases compared to 4.6% for CoT, with both showing similar timeout rates around 11.0%. Performance improved notably with few-shot CoT, which achieved a 21.1% average rate of returned results - the second-highest across all methods. Among all approaches, LtM delivered the best endpoint performance, achieving the highest average rate of successful result returns at 26.3%. In general, a trend is observed from CoT, few-shot CoT, CP-A to LtM, where the share of empty results steadily decreased from 73.7% to 25.0%.

For the Content-Based Evaluation, only those instances

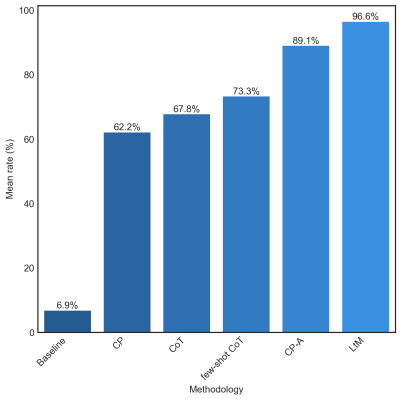


Fig. 4. **Rate of syntactic valid SPARQL queries for evaluated methodologies.**

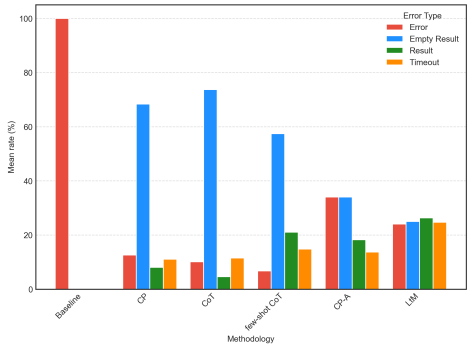


Fig. 5. **Endpoint result distribution for syntactic valid SPARQL queries.**

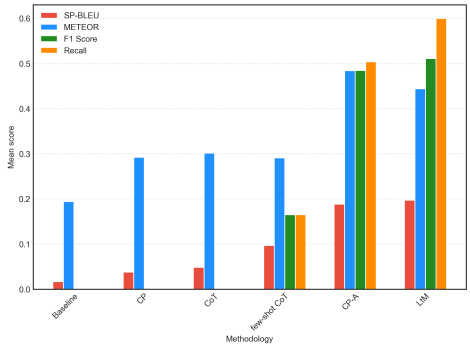


Fig. 6. **Approximated performance metrics by methodology, excluding empty results and query errors.**

were considered where queries returned non-empty results. It is important to note that all reported mean F1, precision, and recall scores in this section are approximations, calculated over successful executions. As the Baseline produced no successful results (Figure 5), its F1, precision, and recall scores are consequently 0.0. Similarly, both CP and CoT, despite yielding limited results rates of 8.0% and 4.6% respectively, achieved no measurable performance under Content-Based Evaluation (i.e., all scores remained at 0.0 in Table II). In contrast, few-shot CoT, CP-A and LtM, which returned results more consistently, revealed a clear upward trend in performance (Figure 6). Few-shot CoT, with an average returned results for 21.1% of queries, achieved an F1-score of 0.16. CP-A slightly lagged in the success rate of returning results (18.2%) but outperformed few-shot CoT with an F1 score of 0.48. The best performance was observed with LtM, which achieved the highest returned result rate (26.3%) and the highest F1 score of 0.51, confirming its effectiveness in generating accurate and semantically aligned federated SPARQL queries compared to all other prompting methodologies.

As illustrated in Figure 6, there is a general positive correlation between the mean F1 scores and the corresponding SP-BLEU score, indicating that structural similarity often aligns with result accuracy. Figure 7 reports the distribution of F1 score across 10 experimental runs for CP-A, few-shot CoT, and LtM. Notably, LtM presents the most consistent performance, with F1 score tightly clustered with a narrow interquartile range (0.49 to 0.57), suggesting greater stability. This observation is further supported by Figure 8, which plots the mean F1 scores along with their 95% confidence interval. The results

TABLE II
APPROXIMATED PERFORMANCE METRICS BY METHODOLOGY, EXCLUDING EMPTY RESULTS AND QUERY ERRORS

Methodology	SP-BLEU	METEOR	F1 Score	Recall	Precision
Baseline	0.02	0.19	0.00	0.00	0.00
CP	0.04	0.29	0.00	0.00	0.00
CoT	0.05	0.30	0.00	0.00	0.00
few-shot CoT	0.10	0.29	0.16	0.16	0.16
CP-A	0.19	0.48	0.48	0.50	0.49
LtM	0.20	0.44	0.51	0.60	0.49

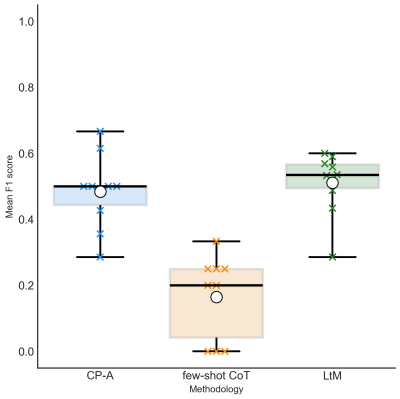


Fig. 7. **Approximated F1 scores for CP-A, few-shot CoT and LtM, excluding empty results and query errors.**

show that few-shot CoT has the highest variability, followed by CP-A, whereas LtM maintains the tightest confidence interval, indicating its robustness and stability across multiple runs.

Table III provides a detailed breakdown of the mean prompt tokens, completion tokens, and total tokens consumed by each strategy. While the Baseline, CoT, and CP show modest increases in total token usage, a significant increase is seen from

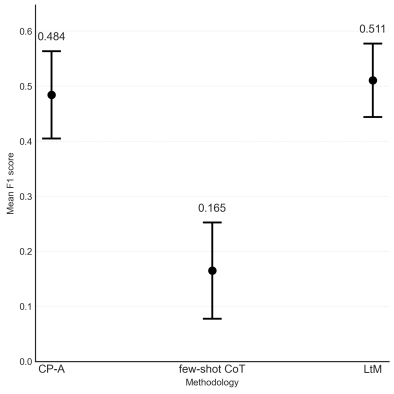


Fig. 8. **Approximated F1 scores with 95% Confidence Intervals, excluding empty results and query errors.**

TABLE III
TOKEN USAGE BY METHODOLOGY

Methodology	Prompt Tokens	Completion Tokens	Total Tokens
Baseline	1,849.95	120.47	1,970.41
CoT	2,320.91	309.69	2,630.60
CP	2,347.88	243.92	2,591.80
CP-A	5,766.64	167.27	5,933.92
few-shot CoT	15,968.13	820.18	16,788.32
LtM	24,301.40	709.58	25,010.99

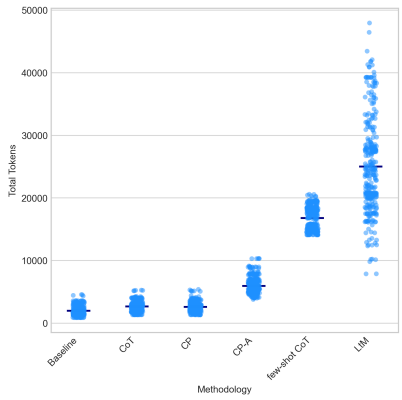


Fig. 9. **Distribution of total used tokens by methodology.**

CP (2,592 tokens) to CP-A (5,933 tokens). The most resource-intensive strategies are few-shot CoT and LtM, demanding 16,788 and 25,011 total tokens on average, respectively. Figure 9 provides information about the variance of average tokens used by methodology. While all methodologies provide a rather dense variance, LtM shows a significant spread in variance, ranging from approximately 8,000 to 48,500 total tokens.

Figure 10 presents the ratio of prompt to completion tokens for each methodology, illustrating their proportional composition. It also includes the corresponding mean F1 scores to facilitate an assessment of their correlation and effectiveness. The figure underlines that for the more complex strategies, like few-shot CoT and LtM, prompt tokens constitute the majority of the total token count. For instance, in the LtM approach,

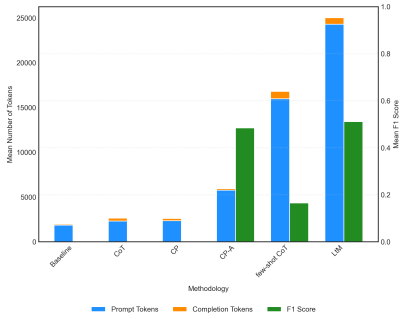


Fig. 10. **Prompt and completion token distribution with corresponding approximated F1 scores by methodology.**

prompt tokens (24,301) are vastly greater than completion tokens (710). This imbalance is indicative of the extensive contextual information, detailed instructions, and multi-step reasoning frameworks provided to the LLM in the prompt for these methods. The results indicate a general trend where methodologies with higher prompt token usage (CP-A, few-shot CoT, LtM) tend to achieve better F1 scores compared to those with minimal token input. However, it is notable that CP-A achieves the highest performance without necessitating the extensive token usage observed for few-shot CoT or LtM.

VI. Discussion

Results demonstrate varying effectiveness of prompting methodologies for federated SPARQL query generation over biomedical KGs. Beginning with Execution-Based Evaluation, the Baseline rarely produced syntactically valid queries. In contrast, providing explicit instructions (as in CP) significantly improved syntactic validity, which was further enhanced by providing the LLM with relevant example question-SPARQL query pairs. By providing the LLM with relevant examples (CP-A) shows a better performance than providing static chain-of-thought few shots (few-shot CoT). This aligns with prior findings on the importance of example sensitivity in prompting [13]. While for few-shot CoT examples the highest similarity against SPARQL queries of the evaluation set was 0.75 (Figure 2), the similarity of all queries against each other was 0.92 (Appendix B). As queries of the evaluation set are also part of the example question-SPARQL query pairs, these findings demonstrate that contextually relevant examples substantially benefit federated SPARQL query generation.

The endpoint behavior revealed that evaluating syntactic correctness alone is insufficient. Although CoT and few-shot CoT resulted in the lowest error rates at the endpoints, they frequently returned empty results, indicating insufficient schema and ontology grounding. Few-shot CoT mitigated this to some extent by providing more valid results than CoT. LtM outperformed all methods, achieving both the lowest rate of empty results and the highest rate of successful query returns. This strongly suggests its decomposition strategy supports more robust and semantically aligned SPARQL query construction.

This superior capability of LtM was further validated when evaluating the quality of the returned results using the Content-Based Evaluation. Specifically, higher F1 (0.51) and recall (0.60) scores demonstrate this: its comparable strong recall score shows an ability to retrieve a larger portion of all correct information, while its leading F1 score, reflects the superior overall accuracy and completeness. LtM and CP-A share identical precision (0.49), indicating that while the accuracy of items retrieved by both methods is comparable, LtM's superior recall (0.60 to 0.50) enables it to capture a greater volume of correct information, thus yielding more correct retrieved results overall.

Additionally, SP-BLEU and METEOR achieved significantly higher scores for CP-A and LtM, indicating a closer structural similarity between the predicted and ground truth

queries. Together with their leading mean F1, precision, and recall, these results highlight better schema alignment and the positive effect of contextual enrichment with relevant examples. Still, LtM outperformed CP-A due to its more sophisticated composition strategy. Looking at the more clustered mean F1 scores for LtM further supports the capabilities of a much more reliable federated SPARQL generation that ultimately makes fewer mistakes.

Alongside performance metrics, resource consumption is crucial for overall effectiveness. Strategies such as few-shot CoT and LtM are inherently more resource-intensive due to their lengthy guidance examples and multi-step construction, respectively. While LtM exhibited the highest average total token usage (Figure 9), it also resulted in significantly greater token usage variance, stemming from its dynamic construction. Consequently, LtM's total token usage occasionally fell below that of few-shot CoT and even CP-A, making it less consistent. This dynamic token consumption suggests that LtM might exhibit greater token efficiency for less complex SPARQL queries. This, however, remains unverified and highlights a limitation of this work: Manual inspection indicated that most queries of the evaluation set were highly complex, but without an empirical complexity analysis, no link between generation performance and SPARQL query complexity could be assessed.

On average LtM consumed fewer completion tokens than few-shot CoT, because few-shot CoT's extensive guidance inflates completion tokens. Given that completion tokens are generally more expensive than prompting tokens, this suggests potential cost-effectiveness over few-shot CoT, despite its greater average total token usage. Combined with its superior performance, this makes LtM the most advantageous methodology.

A further limitation is the absence of explicit schema information provided directly to the LLM. While the LLM could implicitly infer some understanding from the example question-SPARQL query pairs, this indirect approach to schema understanding is inherently constrained. The selective scope of entity URIs indexed for the RAG component presents a further limitation. As our indexing focused on a subset of URIs, found to be most important, some relevant entities may not have been included. Consequently influencing the performance of all evaluated methodologies.

More over this work is constraint by the 10-minute query execution timeout, which is detailed in the experiment settings. This limit for generated queries was implemented for experimental management purposes. However, since cached results of ground truth queries were used due to their potentially long execution times, this 10-minute cutoff may have prematurely terminated some generated queries. This could lead to an underestimation of the true performance of the evaluated methodologies, particularly for complex queries that inherently require longer execution times.

VII. Conclusion

This study evaluated prompting strategies for federated SPARQL query generation from natural language over biomedical KGs.

The first research question addressed the effectiveness of Chain-of-Thought prompting. Findings revealed that CoT showed good performance in generating syntactic sound SPARQL queries with lowest average error rates at endpoints, indicating that for structurally composition of federated SPARQL queries CoT is effective. However, it showed difficulties in generating semantically correct queries, indicating insufficient schema and ontology grounding. Furthermore, this work found that explicit instructions, combined with a richer contextual understand outperformed CoT and few-shot CoT. Thus, despite being more effective than the baseline, careful prompt engineering with contextually relevant examples substantially benefit federated SPARQL generation.

Secondly, this study assessed whether LtM shows a better performance in generating accurate federated SPARQL queries than CoT. Among the evaluated prompting strategies, LtM outperformed all other evaluated methodologies, achieving the highest syntactical validity of 96.2%, the lowest rate of empty results, scoring best in the Query Structure Similarity Evaluation, and having the highest content-based performance. While CP-A also showed a comparable performance, LtM's superority stems from its decompositional approach. It allows for more robust generation of semantically coherent and executable federated SPARQL queries.

The third research question considered whether the accuracy improvements from LtM justify its higher computational cost compared to CoT. LtM was indeed the most token-intensive approach, averaging 25,011 total tokens per query, substantially more than few-shot CoT (16,788) and vanilla CoT (2,630) (Table III). However, LtM consumed fewer completion tokens (710) on average than few-shot CoT (820), a relevant factor given the often higher cost of completion tokens. Furthermore, LtM exhibited considerable variance in total token usage, suggesting potential for greater efficiency with less complex queries due to its dynamical approach. Given its superior accuracy, reliability, and ability to generate semantically meaningful results, LtM's benefits justify its increased resource demand. Particularly for complex federated queries in the biomedical domain where precision and successful data retrieval are substantial.

Future research should explore further refinements of LtM, paired with more comprehensive contextual support about schema and ontology. Providing the LLM with information about Shape Expression Format (ShEx) could be a possible approach to provide a richer understanding of schemas. It should also be further investigated how query complexity impacts federated SPARQL query generation, as this relationship has a substantial influence on overall system performance.

References

[1] H. Zafar, G. Napolitano, and J. Lehmann, "Formal query generation for question answering over knowledge bases," in *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2018, p. 714–728. [Online]. Available: https://doi.org/10.1007/978-3-319-93417-4_46

[2] S. Hu, L. Zou, J. X. Yu, H. Wang, and D. Zhao, "Answering natural language questions by subgraph matching over knowledge graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 5, pp. 824–837, 2018.

[3] Y. Chen, H. Li, G. Qi, T. Wu, and T. Wang, "Outlining and filling: Hierarchical query graph generation for answering complex questions over knowledge graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 8, pp. 8343–8357, 2023.

[4] M. R. A. H. Rony, U. Kumar, R. Teucher, L. Kovriguina, and J. Lehmann, "Sgpt: A generative approach for sparql query generation from natural language questions," *IEEE Access*, vol. 10, pp. 70 712–70 723, 2022.

[5] V. Emonet, J. Bolleman, S. Duvaud, T. M. de Farias, and A. C. Sima, "Llm-based sparql query generation from natural language over federated knowledge graphs," 2025. [Online]. Available: https://arxiv.org/abs/2410.06062

[6] R. Omar, O. Mangukiya, P. Kalnis, and E. Mansour, "Chatgpt versus traditional question answering for knowledge graphs: Current status and future directions towards knowledge graph chatbots," 2023. [Online]. Available: https://arxiv.org/abs/2302.06466

[7] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. Le, and E. Chi, "Least-to-most prompting enables complex reasoning in large language models," 2023. [Online]. Available: https://arxiv.org/abs/2205.10625

[8] J. Bolleman, V. Emonet, A. Altenhoff, A. Bairoch, M.-C. Blatter, A. Bridge, S. Duvaud, E. Gasteiger, D. Kuznetsov, S. Moretti, P.-A. Michel, A. Morgat, M. Pagni, N. Redaschi, M. Zahn-Zabal, T. M. de Farias, and A. C. Sima, "A large collection of bioinformatics question-query pairs over federated knowledge graphs: methodology and applications," 2024. [Online]. Available: https://arxiv.org/abs/2410.06010

[9] J. C. Rangel, T. M. de Farias, A. C. Sima, and N. Kobayashi, "Sparql generation: an analysis on fine-tuning openllama for question answering over a life science knowledge graph," 2024. [Online]. Available: https://arxiv.org/abs/2402.04627

[10] M. Arazzi, D. Ligari, S. Nicolazzo, and A. Nocera, "Augmented knowledge graph querying leveraging llms," 2025. [Online]. Available: https://arxiv.org/abs/2502.01298

[11] N. Tufek-Oezkaya, B. Y. Uslu, V. P. Just, T. Bandyopadhyay, A. S. Thuluva, M. Sabou, and A. Hanbury, "A new retrieval-augmented generation (rag) approach for querying and constructing large-scale knowledge graphs."

[12] L.-P. Meyer, J. Frey, F. Brei, and N. Arndt, "Assessing sparql capabilities of large language models," 2025. [Online]. Available: https://arxiv.org/abs/2409.05925

[13] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023. [Online]. Available: https://arxiv.org/abs/2201.11903

[14] H. M. A. Zahera, A. Manzoor, M. Sherif, D. Moussallem, and A.-C. Ngonga Ngomo, "Generating sparql from natural language using chain-of-thoughts prompting," in *SEMANTiCS*, 2024.

[15] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," 2023. [Online]. Available: https://arxiv.org/abs/2205.11916

[16] M. Saleem, A. Hasnain, and A.-C. Ngonga Ngomo, "Largerdfbench: A billion triples benchmark for sparql endpoint federation," *Journal of Web Semantics*, vol. 48, pp. 85–125, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570826817300719

[17] T. Z. Zhao, E. Wallace, S. Feng, D. Klein, and S. Singh, "Calibrate before use: Improving few-shot performance of language models," 2021. [Online]. Available: https://arxiv.org/abs/2102.09690

[18] R. Q. Dividino and G. Gröner, "Which of the following sparql queries are similar? why?" in *LD4IE@ ISWC*, 2013.

[19] M. Renze, "The effect of sampling temperature on problem solving in large language models," in *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, 2024, p. 7346–7356. [Online]. Available: http://dx.doi.org/10.18653/v1/2024.findings-emnlp.432

[20] N. Goyal, C. Gao, V. Chaudhary, P.-J. Chen, G. Wenzek, D. Ju, S. Krishnan, M. Ranzato, F. Guzmán, and A. Fan, "The Flores-101 evaluation benchmark for low-resource and multilingual machine translation," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 522–538, 2022. [Online]. Available: https://aclanthology.org/2022.tacl-1.30/

[21] S. Banerjee and A. Lavie, "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments," in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, J. Goldstein, A. Lavie, C.-Y. Lin, and C. Voss, Eds. Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 65–72. [Online]. Available: https://aclanthology.org/W05-0909/

APPENDIX

A. Column Matching Test Set

TABLE IV
TEST SET FOR COLUMN NAME MATCHING EVALUATION

Ground Truth Column	Predicted Column
Organism	species
human	homo sapiens
compound	drug
gene_name	gene
target	protein target
clinical phase	development stage
uniprot id	id
uniprotName	uniprot
patentURI	patent
swisslipid	lipid

B. Evaluation Set Levenshtein Similarity Scores

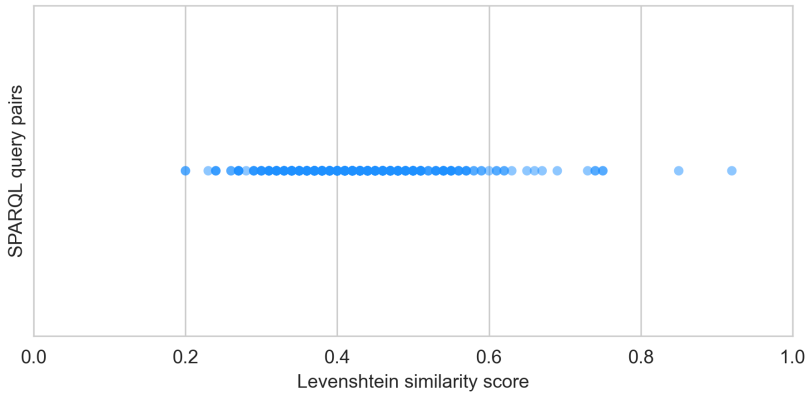


Fig. 11. **Levenshtein similarity scores for all SPARQL query pairs of the evaluation set.**

C. Entity Retrieval Evaluation

TABLE V
RETRIEVAL PERFORMANCE ON THE CHEBI / UNIPROT SUBSET (APPROXIMATELY 620,000 ITEMS).

Model	MRR@5	MRR@10	P@1	NDCG@5	Recall@5	Recall@10
Dense (BAAI/bge-small-en-v1.5)	0.952^{b}	0.952^{b}	$0.952^{b,c}$	0.953^{b}	0.953^{b}	0.953
Sparse (Splade_PP_en_v1)	0.899	0.901	0.869	0.910	0.943	0.959
Hybrid (50/50 weighting)	$0.958^{a,b}$	$0.958^{a,b}$	0.924^{b}	$0.968^{a,b}$	$0.997^{a,b}$	$0.997^{a,b}$

Superscripts mark statistically significant differences ($\alpha = 0.05$, two-sided paired Student's t-test)

D. System Prompt

```
You are a SPARQL query assistant that helps users to create SPARQL queries from natural language
questions to navigate the resources and databases from mainly the Swiss Institute of Bioinformatics and
other biomedical resources.
```

E. Extraction Prompt

```
Given a user question extracts the following:

- **High level concepts** and **potential classes** that could be found in the SPARQL endpoints and used
to     answer the question.
- **Potential entities** and instances of classes that could be found in the SPARQL endpoints and used to
answer the question.
- Split the question in standalone smaller parts that could be used to build the final query (if the
question   is already simple enough, you can return just 1 step).
```

F. Baseline

```
\UseRawInputEncoding

Potential entities extracted from the user question: {{potential_entities}}

Potential classes extracted from the user question: {{potential_classes}}

**Task:**
Generate a federated SPARQL query to answer the user question, utilizing the provided inputs (question,
classes, entities).

**Requirements:**

1.**Output Format:** Provide the generated SPARQL query inside a single markdown code block with the
"sparql" language tag (````sparql ... ````).

2. **Endpoint Comment:** - The *very first line* inside the code block *must* be a comment containing the
URL of the *primary* SPARQL endpoint through which the federated query should
be initiated.
                          - Include only this single primary endpoint URL comment at the start. No other
                          text on this line or preceding it within the code block.
```

G. Construction Prompt

```
Potential entities extracted from the user question {{potential_entities}}

Potential classes extracted from the user question  {{potential_classes}}

The following are SPARQL endpoint descriptions where the federated sparql query can be executed on or can
federate with:

---
UniProt SPARQL
https://sparql.uniprot.org/sparql

A comprehensive endpoint containing triples of protein-related data, providing access to protein
sequences, functional annotations, and cross-references, making it essential for querying protein
information and their biological functions.
---
Rhea DB SPARQL
https://sparql.rhea-db.org/sparql

An expert-curated endpoint containing biochemical reactions that uses ChEBI ontology for chemical
entities, allowing users to query detailed information about enzymatic reactions, transport reactions,
and spontaneous reactions in biological systems.
---
SwissLipids SPARQL
https://sparql.swisslipids.org/sparql/

A specialised endpoint containing lipid structures that provides detailed information about lipid
biology, including lipid classifications, metabolic reactions, and associated enzymes, making it valuable
for lipidomics research and lipid-related queries.
---

**Task:**
Generate a **federated SPARQL query** to answer the user question, utilising the provided inputs
(question, classes, entities, endpoints).

**Requirements:**

1.**Answer Question:** The query must aim to retrieve information that directly answers the user's
question.
```

```
2.**Federated:** The query **must** use `SERVICE <endpoint_uri>  ...` clauses to query the relevant
endpoints from the provided context list as needed.

3.**Prefixes:** For better readability, the query **should** contain prefixes.

4.**Computational Efficiency:** As a guideline, aim for computational efficiency by keeping the query
structure as simple and direct as possible to answer the question.

5.**Comments:** Provide comments in the query to explain the logic behind the construction and any
assumptions made during the generation process.

6.**Output Format:** Provide the generated SPARQL query inside a single markdown code block with the
"sparql" language tag (````sparql ... ````).

7. **Endpoint Comment:** - The *very first line* inside the code block *must* be a comment containing the
URL                            of the *primary* SPARQL endpoint (provided in the input context) through
which                                 the federated query should be initiated.

                           - Include only this single primary endpoint URL comment at the start. No other
                           text    on this line or preceding it within the code block.

8. **Context Adherence:** - Always derive your answer from focusing on endpoint URLs and schema
information                                    provided in the input context. Do not try to create a query
from nothing and do                                    not provide a generic query.

                            - Due to the nature of federated sparql queries it might be that some are not
                            included in the input context. In this case reason about for which endpoints
                            (databases) it would make sense to federate with.
```

H. Augmented Construction Prompt

```
Potential entities extracted from the user question {{potential_entities}}

Potential classes extracted from the user question  {{potential_classes}}

Example  queries (**Note:** These queries are examples of a natural questions with a corresponding sparql
query. They may be not federated, but give hints about how the schema of an endpoint look like)
{{extracted_example_queries}}

The following are SPARQL endpoint descriptions where the federated sparql query can be executed on or can
federate with:

---
UniProt SPARQL
https://sparql.uniprot.org/sparql

A comprehensive endpoint containing triples of protein-related data, providing access to protein
sequences, functional annotations, and cross-references, making it essential for querying protein
information and their biological functions.
---
Rhea DB SPARQL
https://sparql.rhea-db.org/sparql

An expert-curated endpoint containing biochemical reactions that uses ChEBI ontology for chemical
entities, allowing users to query detailed information about enzymatic reactions, transport reactions,
and spontaneous reactions in biological systems.
---
SwissLipids SPARQL
https://sparql.swisslipids.org/sparql/

A specialised endpoint containing lipid structures that provides detailed information about lipid
biology, including lipid classifications, metabolic reactions, and associated enzymes, making it valuable
for lipidomics research and lipid-related queries.
---

**Task:**
Generate a **federated SPARQL query** to answer the user question, utilising the provided inputs
(question, classes, entities, endpoints).

**Requirements:**
```

```
1.**Answer Question:** The query must aim to retrieve information that directly answers the user's
question.

2.**Federated:** The query **must** use `SERVICE <endpoint_uri>  ...` clauses to query the relevant
endpoints from the provided context list as needed.

3.**Prefixes:** For better readability, the query **should** contain prefixes.

4.**Computational Efficiency:** As a guideline, aim for computational efficiency by keeping the query
structure as simple and direct as possible to answer the question.

5.**Comments:** Provide comments in the query to explain the logic behind the construction and any
assumptions made during the generation process.

6.**Output Format:** Provide the generated SPARQL query inside a single markdown code block with the
"sparql" language tag (````sparql ... ````).

7. **Endpoint Comment:** - The *very first line* inside the code block *must* be a comment containing the
URL                          of the *primary* SPARQL endpoint (provided in the input context) through
which the                          federated query should be initiated.

                        - Include only this single primary endpoint URL comment at the start. No other
                        text    on this line or preceding it within the code block.

8. **Context Adherence:** - Always derive your answer from focusing on endpoint URLs and schema
information                                 provided in the input context. Do not try to create a query
from nothing and do                                 not provide a generic query.

                         - Due to the nature of federated sparql queries it might be that some are not
                         included in the input context. In this case reason about for which endpoints
                         (databases) it would make sense to federate with.
```

I. Few-shot Chain-of-Thought examples

The few-shot Chain-of-Thought (CoT) examples, omitted here due to their extensive length, can be found in the accompanying GitHub repository.[3]

J. Least-to-Most Planning Prompt

```
Identified entities and classes from the user question \{{entities\_question\}} \{{classes\_question\}}

Potential entities extracted from the user question \{{potential\_entities\}}

Potential classes extracted from the user question \{{potential\_classes\}}
Example queries (**Note:** These queries are examples of a natural questions with a corresponding sparql
query. They may be not federated, but give hints about how the schema of an endpoint look like)
\{{extracted\_example\_queries\}}

The following are SPARQL endpoint descriptions where the federated sparql query can be executed on or can
federate with:

---UniProt SPARQLhttps://sparql.uniprot.org/sparql

A comprehensive endpoint containing triples of protein-related data, providing access to protein
sequences, functional annotations, and cross-references, making it essential for querying protein
information and their biological functions.---Rhea DB SPARQLhttps://sparql.rhea-db.org/sparql

An expert-curated endpoint containing biochemical reactions that uses ChEBI ontology for chemical
entities, allowing users to query detailed information about enzymatic reactions, transport reactions,
and spontaneous reactions in biological systems.---SwissLipids
SPARQLhttps://sparql.swisslipids.org/sparql/

A specialised endpoint containing lipid structures that provides detailed information about lipid
biology, including lipid classifications, metabolic reactions, and associated enzymes, making it valuable
for lipidomics research and lipid-related queries.---

**Task:**

- **Map identified entties or classes from the user question to ontology IRIs that are needed to build
the sparql query** (If there is not a good match in the provided context (potential entities, potential
classes, example queries) figure out a mapping from your knowledge)
```

[3]https://github.com/sebastians1999/federated-sparql-rag-agent.git

```
{ Determine return labels (early draft `SELECT` vars)

- Locate **SPARQL endpoints** we will need to answer the question and what information we seek to
retrieve from that endpoint.

{ Choose the **primary endpoint** where the query is going to be executed.

**Example Output:**

QueryPlan("iri\_map": \{"Alzheimer's Disease": "http://example.org/ontology/Disease\_123","APP gene":
"http://example.org/ontology/Gene\_456"\},"early\_select": ["?disease", "?gene"],"federated\_endpoints":
["https://sparql.rhea-db.org/sparql","https://sparql.uniprot.org/sparql"],"target\_endpoint":
"https://sparql.uniprot.org/sparql")
```

K. Least-to-Most Service Pattern Prompt

```
**Task**

You are writing **one remote SERVICE block** for a federated SPARQL query.
For the endpoint shown in *Endpoint for this block* you must output **exactly one** self-contained
snippet that:

* lists every `PREFIX` required by the snippet,
* wraps the triple pattern(s) inside **`SERVICE <{{service_endpoint}}> { ... }`**,
* is itself wrapped in `ASK WHERE { ... }` so we can test it quickly.

The block will not answer the user question on its own; its job is to fetch **partial information** that
the primary endpoint (`{{target_endpoint}}`) will later join with other blocks.
Be reasonable about the amount and granularity of data you pull from this remote endpoint.

---

**Requirements**

1. **Exactly one block** { shape:
   `PREFIX ... ASK WHERE { SERVICE <{{service_endpoint}}> { ... } }`
2. **Use the IRI bindings** given in *Mapping* exactly (or IRIs from the examples); do not invent IRIs.
3. **Keep it minimal but meaningful**
   *Prefer the shortest predicate path that is likely to work; add expansions like `rdfs:subClassOf*` only
   when necessary.*
4. **Variable names**
   *Use descriptive names (`?lipid`, `?enzyme`, ...). You may use `?x` once as a generic placeholder.*
5. **Learn from previous failures**. Consult the error message in *Failed patterns* if one is provided.
6. **Look at examples** to understand how the schema looks like and what predicates/paths can be used.

---

**Output format**

* Provide the generated SPARQL query inside a single markdown code block with the "sparql" language tag
(````sparql ... ````).

* The *very first line* inside the code block *must* be a comment containing the URL of the *primary*
SPARQL endpoint (provided in the input context) through which the federated query should be initiated.

<example start>
PREFIX rh: <http://rdf.rhea-db.org/>

ASK WHERE {
  SERVICE <https://sparql.rhea-db.org/sparql> {
    ?rhea      rh:side/rh:contains/rh:compound ?compound .
    ?compound  (rh:chebi
               | rh:reactivePart/rh:chebi
               | rh:underlyingChebi/rh:chebi) ?metabolite .
  }
}
<example end>

**User question:** {{input}}

{{endpoint_description}}
```

```
**Example queries:** (natural question <-> SPARQL pairs; may not be federated)
{{extracted_example_queries}}

**Endpoint for this block (remote SERVICE endpoint):** {{service_endpoint}}

**Target endpoint** (primary execution endpoint): {{target_endpoint}}

**Mapping of entities identified in user question to ontology IRIs:** {{iri_mapping}}

**Failed patterns:** {{failed_pattern}}
```

L. Least-to-Most Primary Pattern Prompt

```
**Task**

You are writing **the primary (local) block** of a federated SPARQL query.
This block executes on the *primary* endpoint (`{{target_endpoint}}`), so **do not** wrap it in a
`SERVICE` clause. Produce **exactly one** self-contained snippet that:

* lists all necessary `PREFIX` declarations,
* places the triple pattern(s) directly inside the query body,
* is wrapped in `ASK WHERE { ... }` so we can test it quickly.

The block alone will not fully answer the user's question; its results will later be joined with remote
blocks.

---
**Requirements**

1. **Exactly one block** { shape:
   `PREFIX ... ASK WHERE { ... }` (no `SERVICE` clause)
2. **Use the IRI bindings** in *Mapping* exactly (or IRIs from the examples); do not invent IRIs.
3. **Keep it minimal but meaningful**
   *Choose the shortest predicate path likely to work; add limited expansions such as `rdfs:subClassOf*`
   only when essential.*
4. **Variable names**
   *Use descriptive names (`?lipid`, `?enzyme`, ...). You may use `?x` once for a generic placeholder.*
5. **Learn from previous failures**. Consult the error message in *Failed pattern* if one is provided.
6. **Look at examples** to understand how the schema looks like and what predicates/paths can be used.

---
**Output format**

* Provide the generated SPARQL query inside a single markdown code block with the "sparql" language tag
(````sparql ... ````).

* The *very first line* inside the code block *must* be a comment containing the URL of the *primary*
SPARQL endpoint (provided in the input context) through which the federated query should be initiated.

<example start>
PREFIX rh: <http://rdf.rhea-db.org/>

ASK WHERE {
  ?rhea      rh:side/rh:contains/rh:compound ?compound .
  ?compound  (rh:chebi
             | rh:reactivePart/rh:chebi
             | rh:underlyingChebi/rh:chebi) ?metabolite .
}
<example end>

**User question:** {{input}}

{{endpoint_description}}

**Example queries:** (natural-language question <-> SPARQL answer pairs; may not be federated)
{{extracted_example_queries}}

**Target endpoint** (primary execution endpoint): {{target_endpoint}}

**Mapping of entities identified in user question to ontology IRIs:** {{iri_mapping}}

**Failed patterns:** {{failed_pattern}}
```

M. Least-to-Most Assemble Prompt

```
**Task**
Your task is to merge validated building blocks into **one runnable federated query**.
The query needs to be created for the provided **target endpoint**.

Assemble a complete SPARQL query that:

1. **Runs on** the target endpoint: {{target_endpoint}}
   This endpoint is the host; all other calls go inside `SERVICE {}` blocks.

2. **Federates with** the following remote endpoints:{{federated_endpoints}}

3. **Integrates** the **validated triple patterns** listed below, keeping their semantics intact but
adjusting them as necessary (e.g., renaming variables, harmonizing prefixes) so they fit together into a
single coherent query.

4. It might happen that for some endpoints none of the triple patterns/building blocks are working
(indicated with <failed pattern>, <successful pattern>).**Learn from these previous failures** and
orientate yourself on provided examples and context.

5. **Identify** join variables between endpoints.

6. Incorporates the *early draft* 'SELECT' variables. These have been used at an earlier stage as a
'north star' to serve as a rough orientation for the generation of the building blocks. It might be
necessary to change these or include additional variables.

7. Include all **PREFIX** that are needed for the final query.

**Output format**

* Provide the generated SPARQL query inside a single markdown code block with the "sparql" language tag
(````sparql ... ````).

* The *very first line* inside the code block *must* be a comment containing the URL of the *primary*
SPARQL endpoint (provided in the input context) through which the federated query should be initiated.

**User question:** {{input}}

**Example queries:** (**Note:** These queries are examples of a natural questions with a corresponding
sparql query. They may be not federated, but give hints about how the schema of an endpoint look like)

{{extracted_example_queries}}

**Building blocks generated from the user question, examples and IRI mapping:**

{{building_blocks}}

**Mapping of entities identified in user question to ontology IRIs:**

{{iri_mapping}}
```