Generating SPARQL from Natural Language Using Chain-of-Thoughts Prompting

Hamada M. ZAHERA, Manzoor ALI, Mohamed Ahmed SHERIF, Diego MOUSSALLEM, and Axel-Cyrille Ngonga NGOMO

Abstract.
Purpose: SPARQL is a highly expressive query language for knowledge graphs; yet, formulating precise SPARQL queries can be challenging for non-expert users. A potential solution is translating natural questions into SPARQL queries, known as SPARQL generation. This paper addresses the challenges of translating natural language questions into SPARQL queries for different knowledge graphs.

Methodology: We propose COT-SPARQL, our approach to generate SPARQL queries from input questions. Our approach employs Chain-of-thoughts prompting that guides large language models through intermediate reasoning steps and facilitates generating precise SPARQL queries. Furthermore, our approach incorporates entities and relations from the input question, and one-shot example in the prompt to provide additional context during the query generation process.

Findings: We conducted several experiments on benchmark datasets and showed that our approach outperforms the state-of-the-art methods by a large margin. Our approach achieves a significant improvement in F1 score of 4.4% and 3.0% for the QALD-10 and QALD-9 datasets, respectively.

Value: Our COT-SPARQL approach contributes to the semantic web community by simplifying access to knowledge graphs for non-expert users. In particular, COT-SPARQL enables non-expert end-users to query knowledge graphs in natural languages, where COT-SPARQL converts user natural languages queries into SPARQL queries, which can be executed via the knowledge graph’s SPARQL endpoint.

Keywords: SPARQL Generation, Large Language Models, Chain-of-Thoughts.

1. Introduction

Knowledge graphs (KGs) are valuable sources of structured information that can be queried using SPARQL, a standard query language for the Semantic Web. However,
SPARQL is an expressive language that requires users to have a deep knowledge of its syntax and semantics, as well as the specification of knowledge graph’s schema [1–3]. This poses challenges for non-expert users to formulate and execute SPARQL queries, consequently limiting the accessibility and usability of knowledge graphs. To mitigate these challenges, SPARQL generation, has emerged as an active research area to bridge the gap between natural language and SPARQL [4].

SPARQL generation is the task of automatically converting natural language questions into SPARQL queries (e.g., see Figure 1), which can be executed over knowledge graphs. Current methods for SPARQL generation involve several challenges, such as mapping natural language terms to their corresponding entities and relations in knowledge graphs [5]. For example, template-based methods for SPARQL generation often require multiple steps and are tailored to specific knowledge graphs, which limit their applicability across different systems [6]. These challenges increase significantly when dealing with large and diverse knowledge graphs, such as Wikidata. Additionally, machine learning approaches for SPARQL generation aim to learn mapping and transformation rules from a large corpus of question-SPARQL pairs [7, 8]. However, these approaches require huge annotated data to train effective models. Recently, large language models (LLMs) have shown remarkable capabilities in generating database queries, such as SQL, from natural questions [9, 10]. However, generating SPARQL queries from a natural language question is more challenging, as it: 1) involves mapping natural language terms to entities and relations in knowledge graphs, and 2) requires constructing complex queries that match the semantics of questions.

To address these challenges, we leverage Chain-of-Thoughts Prompting (CoT), which has been shown to elicit the reasoning skills of LLMs for various tasks with few-shot examples [11]. We propose CoT-SPARQL, our approach for SPARQL generation based on Chain-of-Thoughts reasoning. Our approach guides LLMs to think step-by-step and generate SPARQL queries similar to given few-shot examples. Unlike existing methods that rely on pre-defined templates or fixed rules, our approach can dynamically adapt to different knowledge graphs, and generate queries that capture natural language semantics. We conducted several experiments on benchmark datasets and evaluated the performance against several baselines. Our approach outperforms state-of-the-art methods in terms of
accuracy, error-free SPARQL queries on several benchmark datasets. We summarize the main contributions of our paper as follows:

- We propose a new approach for SPARQL generation from natural questions using Chain-of-Thoughts prompting.
- Unlike existing methods, our approach adapts to different questions and KGs, and generates queries that capture the complex and diverse semantics of natural language.
- We show that our approach outperforms the state-of-the-art baselines on different datasets. Our implementation is open source and publicly available.

2. Related Works

2.1. SPARQL Generation

Generating SPARQL queries is an essential task for accessing and analyzing Semantic Web data. Previous studies have primarily focused on two directions: manual- and schema-based SPARQL generation. In manual approaches, human experts create SPARQL queries to test ontology systems [12, 13] or to identify query features from existing datasets [14–16]. However, these approaches are not scalable to large and dynamic knowledge graphs such as Wikidata [17], which require a diverse set of queries to cover various aspects of the data. In contrast, schema-based approaches automatically generate SPARQL queries from pre-defined schemas or templates, which can overcome the limitations of manual approaches [18–20]. Such schemas define the structure and semantics of queries and use rules to insert data values from knowledge graphs into the queries. While these methods have shown promising results in generating complex and diverse queries [20–22], but they rely on a pre-defined set of templates, which limits the variety and scope of the queries. Moreover, creating new schemas for different question types involves manual effort, which reduces the scalability and automation of SPARQL generation process.

Another research direction has investigated the use of neural machine translation for SPARQL generation. For instance, Soru et al. [23] presented a sequence-to-sequence model that learns to generate SPARQL patterns from natural language questions. The authors used a semi-supervised approach with pre-defined templates to align questions and queries, and train their model on large-scale knowledge graphs. This approach can generate complex queries that involve multiple graph patterns, but also requires a lot of training data. Moreover, Zafar et al. [24] developed a method called SQG, which generates SPARQL queries from large-scale knowledge graphs. The proposed method has a modular design to integrate with other question answering components. Notably, this method can handle questions that are noisy or complex by finding a minimal sub-graph. However, this method encounters several challenges, such as handling out-of-vocabulary words, generalizing to unseen questions, and finding relevant query patterns.

On the other hand, Rony et al. [25] proposed the SGPT model that converts natural questions into SPARQL queries. SGPT is a comprehensive approach that does not depend on specific knowledge graphs or manual query templates. Specifically, SGPT leverages the GPT-2 language model and incorporates both linguistic and graph-specific features.

2https://github.com/dice-group/CoT-Sparql
into its parameters. In contrast to the previous studies, our approach employs LLMs (e.g., LLaMA2-Code) to generate SPARQL queries using Chain-of-thoughts prompting without requiring predefined schemas or graph structures.

2.2. Chain-of-Thoughts Prompting

LLMs prompting has significantly improved the performance across various natural language processing tasks [26]. However, recent studies indicate that basic prompts (e.g., “Generate a SPARQL code for the input question”) may not always lead to precise results [11]. Recently, researchers have adopted Chain-of-Thoughts prompt as a means to enhance the capabilities of large language models in reasoning and generating tasks [27]. CoT reflects the step-by-step learning process of humans, methodically moving through stages towards a solution, and leveraging context and supplementary information as necessary to achieve its objectives. Since our study focuses on code generation, we review related studies that apply Chain-of-Thoughts for this purpose. For example, Li et al. [28] leveraged CoT approach in combination with zero-shot and In-context learning to extract specialized coding abilities from large language models. Furthermore, Jiang et al. [29] investigated the application of LLMs for code generation through a CoT-based approach, including planning and implementation steps. Their structured approach demonstrates clear advantages over traditional direct generation methods using language models. Additionally, Pourreza and Rafiei [30] developed a CoT-based approach for text-to-SQL generation, achieving a notable improvement of 10% in performance.

For SPARQL generation, Yang et al. [31] proposed an LLM-based approach to generate SPARQL queries for Chinese knowledge graphs. Their method involves prompting an LLM with an input question, including entity mentions and their URIs, followed by the phrase “the SPARQL statement corresponding to the graph is”. However, this approach has limitations such as prompting the LLM without additional context, such as few-shot examples (question and SPARQL pairs), may not be efficient for generating complex SPARQL queries. Furthermore, the authors employed a generic pre-trained LLM, ChatGLM-6B, in contrast, we used a specialized model, LLaMa-Code, which is potentially better suited for tasks involving code and logical form generation. Similarly, Kovriguina et al. [32] introduced the SPARQGen approach, a one-shot prompt method for instructing the GPT-3 model to generate SPARQL queries. Their approach involves a basic LLM prompt, which contains a single example of a question, and its corresponding SPARQL query, instructions to explain the task of SPARQL generation for LLM and a test question. This method only considers a fixed set of questions/SPARQL pairs known as guiding examples. During the experiments, the author randomly selected a guiding example to provide a context for the LLM prompt. However, the randomly-selected example may not be relevant to the input question.

In contrast, our approach consider few-shot examples based on semantic similarity. In particular, we cluster the training set into groups of ⟨question, SPARQL⟩ pairs, then select the most semantically similar example to the input question from the appropriate cluster. Furthermore, SPARQGen approach employs a basic LLM prompt (“Given the following user question and RDF graph ... generate the corresponding SPARQL query...”), our Chain-of-Thoughts prompt incorporates the instruction “Let’s think step by step” which triggers the reasoning capabilities of LLM during token generation, resulting in more precise results [33].
3. Approach

In this section, we present our approach (CoT-SPARQL) for generating SPARQL queries from natural questions, including the components: prompt building, in-context learning and query validation, as shown in Figure 2. CoT-SPARQL starts by introducing the phrase “Let’s think step by step” into the prompt, to enforce structured reasoning capabilities of the LLM is initiated. In our study, we consider LLaMA2-code model due to it’s strong performance in code generation, positioning it as one of the best open-source models for this task [34]. CoT-SPARQL then define the task in the prompt building step (see Section 3.1 for more details), where it providing additional context from the input question, and including a few-shot example. This additional in-context learning helps the LLM better understand the question and generate accurate SPARQL queries (See Section 3.2 for more details). Finally, we verify the correctness of the SPARQL queries prior to their execution (See Section 3.3 for more details).

3.1. Prompt building.

In this component, we use the CoT prompting [35] to enhance the reasoning capabilities of LLMs for Text-to-SPARQL generation. Specifically, we incorporate the phrase “Let’s think step by step” into the LLM prompt to initiate a structured reasoning process and sequentially convert the given question into a SPARQL query.

Figure 3 shows an example run of the prompt building process for the user input query “Was Gerald Gibbs the cinematographer of X the unknown?”.
our prompt building process includes three parts: i) [INST] ... [\INST], where we define the task and its description to the LLM to be SPARQL generation for a target knowledge graph (e.g., DBpedia in Figure 3), ii) Context (A) provides the LLM with additional information such as entities and relations extracted from the input question, and iii) Context (B) presents a few-shot example for generating SPARQL query with the same syntax.

3.2. In-context learning.

In this component, we add two contexts (Context (A) and Context (B) as shown in Figure 2) in our CoT prompt to provide the LLM with additional information for generating precise SPARQL queries.

3.2.1. Context (A)

We enrich the LLM prompt with entities and relations information from the input question. This information helps the LLM to disambiguate entities and understand the intended meaning correctly, thus reducing the chance of getting irrelevant or incorrect results (i.e., hallucination). For this purpose, we preprocess the input question using two libraries: spaCy fishing\(^3\) (for entity linking in Wikidata) and Falcon\(^4\) (for entity linking in DBpedia) and REBEL\(^5\) (for relations extraction). The libraries also handle the issues of prefixes and IRIs in the SPARQL query generation.

3.2.2. Context (B)

We include a one-shot example in our prompt to show the LLM how to convert text to a SPARQL query. This one-shot example contains a (question, SPARQL) pair relevant to the input question. To achieve this, we embed all questions in the training set (i.e., Examples) as semantic vectors using sentence-transformer\(^6\) library. The sentence-transformer captures semantic similarities between sentences, enabling effective clustering of textual data. We apply \textit{K-means} clustering to group similar questions together into clusters. We calculate the cosine similarity to identify the most similar example as one-shot for the input question. We further refine our process by adopting the \textit{K-means++} initialization method\(^7\), which optimizes the selection of initial cluster centers, thereby improving convergence and reducing the likelihood of poor clustering outcomes. Finally, we append the relevant information, such as entities and relations in the SPARQL query, for the selected question as shown in example of Figure 3 in Context B.

3.3. Query validation.

In the final step, we use the SPARQLWrapper library\(^8\) to ensure that the generated SPARQL queries are syntactically correct, avoiding the execution of invalid ones. This library allows for the execution of SPARQL queries on remote endpoints connected to respective

---

\(^3\)https://github.com/Lucaterre/spacyfishing
\(^4\)https://labs.tib.eu/falcon/
\(^5\)https://github.com/Babelscape/rebel
\(^6\)https://github.com/UKPLab/sentence-transformers
\(^7\)We use the Silhouette Score to determine the optimal number of clusters
\(^8\)https://sparqlwrapper.readthedocs.io/en/latest/main.html
4. Experiments

We conducted our experiments to answer the following research questions:

**RQ1.** Does In-context learning enhance the performance of LLMs in generating SPARQL queries?

**RQ2.** How accurate and precise are the the SPARQL queries generated by our approach?

**RQ3.** How does the performance of our approach compare to state-of-the-art approaches of the question answering task?

---

9 https://query.wikidata.org/
10 https://dbpedia.org/sparql
Table 1. Summary of datasets used in our experiments.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>KG</th>
<th>Test</th>
<th>Valid</th>
<th>Train</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>QALD-9</td>
<td>DBpedia</td>
<td>150</td>
<td>58</td>
<td>350</td>
<td>Multilingual</td>
</tr>
<tr>
<td>VQuAnDa</td>
<td>DBpedia</td>
<td>1000</td>
<td>500</td>
<td>3500</td>
<td>English</td>
</tr>
<tr>
<td>LC-QuAD 2.0</td>
<td>Wikidata</td>
<td>5969</td>
<td>2389</td>
<td>21497</td>
<td>English</td>
</tr>
<tr>
<td>QALD-10 2.0</td>
<td>Wikidata</td>
<td>394</td>
<td>-</td>
<td>412</td>
<td>Multilingual</td>
</tr>
</tbody>
</table>

4.1. Datasets.

We used four benchmark datasets in our evaluation, namely: LC-QuAD 2.0, VQuAnDa, QALD-9, and QALD-10. Table 1 shows an overview of the datasets, including number of questions in train, valid and test splits and the language of questions. LC-QuAD 2.0 [36] contains 30k question-query pairs over Wikidata and DBpedia, with 10 categories that vary in complexity and structure, where each question is annotated with its answer type and entities. VQuAnDa [37] has 5k question-query pairs over DBpedia with their verbalised answers, covering different question types, such as Boolean, list, and resource. QALD-9 [38] has 558 total question-query pairs over DBpedia and Wikidata, with temporal, spatial, comparative, superlative, and other reasoning. This dataset supports multilingual question answering over knowledge graphs in 10 languages. QALD-10 [39] has 412 training set question-query pairs over Wikidata, with temporal, spatial, comparative, superlative, and other reasoning. This dataset supports multilingual question answering over knowledge graphs in 9 languages. We used this dataset for our pilot study (see Section 5.3) to evaluate the correctness of generated SPARQL queries in questions answering.

4.2. Baselines.

We compared our approach against different baselines, including state-of-the-art approaches and LLMs with standard prompt (“generate a sparql query for the input question”) as follows:

State-of-the-art approaches:

- SQG [24], which extracts sub-graph patterns from the question and ranks candidate queries by their structural similarity with the question, using a Tree-LSTM model.
- NSpM [23], which trains a Bi-LSTM model with a sequence-to-sequence technique to map natural language questions to template SPARQL queries.
- TeBaQA [40], which predicts the SPARQL query structure from template classes derived from the training dataset, and combines them with a sequence-to-sequence model to generate the SPARQL query.
- SGPT [25] uses a stack of Transformer-encoders to encode linguistic features of natural language questions and a fine-tuned GPT-2 model to decode and generate SPARQL queries.
- SPARQLGen [32], this is the state-of-the-art baseline that prompts the GPT-3 model with a fixed one-shot example to generate SPARQL queries.

LLM baselines with standard prompt\(^\text{11}\):

\(^{11}\text{By standard prompt, we mean to directly prompt the LLM for generating SPARQL queries using natural language questions without neither context learning nor example queries.}\)
• LLaMA2-code [34] is a variant of the LLaMA2 language model specifically designed for code generation tasks. In particular, we used the CodeLlama-Instruct variant, a 34-billion-parameters model as a baseline for generating SPARQL queries from natural language prompts.

• CodeQwen1.5 [41] is a code-specific variant of Qwen1.5 model that has been pre-trained on a large corpus of code data, enabling to handle long context understanding and generation, supporting a context length of up to 64K tokens. Additionally, CodeQwen1.5 offers extensive languages support, supporting a total of 92 coding languages, including SPARQL.

• Mistral-Code [42] is an advanced language model with 7.3B parameters. It is designed to perform on coding tasks, outperforming other models such as Llama-34B in various benchmarks.

4.3. Metrics.

We adopt the same evaluation metrics as in Rony et al. [25] (F1 and BLEU scores) to measure the performance of generating SPARQL queries for three datasets. F1 metric compares the generated SPARQL queries with the gold-standard ones and calculates the harmonic mean of precision and recall. BLEU metric evaluates the generated SPARQL queries based on the similarity of n-grams with one or more gold-standard queries. The BLEU metric is determined through precision and recall at the token level, represented by Equation (1):

\[
BLEU = BP \times \exp \left( \frac{1}{N} \sum_{n=1}^{N} w_n \times \log(p_n) \right)
\]

whereas the F1 score, represented by Equation (2):

\[
F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]

In addition, we used the QALD-specific Macro F1 metric (F1-QALD), designed for evaluating performance over linked data benchmarks [43]. In particular, we employed F1-QALD metric to evaluate the systems performance for the third research question (RQ3). Moreover, F1-QALD metric considers additional semantic information in certain scenarios. If the set of golden answers is not empty and question answering system returns empty set, then precision is set to 1, while recall and F-measure are set to 0.

4.4. Setup and Hardware Requirement

We run our experiments on a server equipped with an AMD EPYC 9334 Processor (64 Threads, 32 cores), 1032GB RAM, and NVIDIA A100 80GB PCIe GPUs. Furthermore, we implemented our approach using Python 3.10 and PyTorch 2.1.1 frameworks. We obtained the pre-trained checkpoints of all models (LlaMA-Code\textsuperscript{12}, Mistral-Code\textsuperscript{13}, CodeQwen1.5\textsuperscript{14}) from the Hugging Face repository.

\textsuperscript{12}https://huggingface.co/TheBloke/CodeLlama-34B-Instruct-GPTQ
\textsuperscript{13}https://huggingface.co/TheBloke/Mistral-7B-codealpaca-lora-GPTQ
\textsuperscript{14}https://huggingface.co/Qwen/CodeQwen1.5-7B-Chat
5. Results and Discussion

5.1. Evaluating the performance of SPARQL generation

In research question (RQ1), we investigate the impact of in-context learning within our COT-SPARQL approach in generating precise SPARQL queries. In particular, we conduct a comparative analysis against different baselines, employing F1 and BLEU as evaluation metrics. As shown in Table 2, our approach (COT-SPARQL) outperforms the baselines on two of three datasets\(^\text{15}\). For example, on the VQuAnDa dataset, COT-SPARQL achieves a BLEU score of 71.61 and an F1 score of 89.36. In contrast, the best Seq2Seq baseline (i.e., SGPT) achieves a BLEU score of 72.58 and an F1 score of 88.87. Moreover, these findings demonstrate that COT-SPARQL, which incorporates entities and relations into the LLM prompt, achieves the highest F1 scores on two datasets (i.e., 89.36 on VQuAnDa and 70.45 on QALD-9) and the second-best performance on the LC-QuAD 2.0 dataset with an F1 score 89.04. These results indicate that COT-SPARQL can effectively leverage the pre-trained knowledge in large language models, and robustly encode the semantic information (e.g., entities and relations) from the input question, to generate accurate SPARQL queries.

5.2. Evaluating the correctness of generated SPARQL queries

To answer RQ2, we reported the number and percentage of valid queries that return correct answers without errors via DBpedia and Wikidata endpoints, and invalid queries that return syntax errors or empty answers, in Table 3. Since the queries generated by the other baselines (NSpM, SQG, TeBaQA, and SGPT) are not publicly available, we were unable to evaluate their validness and only compare our approach with the LLM

\(^{15}\text{The values are obtained from the respective papers: \cite{23,24,32,39}}\)
Table 3. Evaluating the correctness of generated SPARQL queries (RQ2). Best results are in bold.

<table>
<thead>
<tr>
<th>Models</th>
<th>LC-QuAD 2.0</th>
<th>VQuAnDa</th>
<th>QALD-9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valid</td>
<td>Invalid</td>
<td>Valid</td>
</tr>
<tr>
<td>LLaMA2-Code</td>
<td>1216 (25.3%)</td>
<td>3651</td>
<td>759 (75.9%)</td>
</tr>
<tr>
<td>Mistral-Code</td>
<td>121 (2.5%)</td>
<td>4746</td>
<td>46 (4.6%)</td>
</tr>
<tr>
<td>CodeQwen1.5</td>
<td>998 (20.5%)</td>
<td>3869</td>
<td>505 (50.5%)</td>
</tr>
<tr>
<td>COT-SPARQL(_\text{ent})</td>
<td>3243 (75.0%)</td>
<td>1642</td>
<td>951 (95.0%)</td>
</tr>
<tr>
<td>COT-SPARQL(_\text{ent+rel})</td>
<td>4640 (96.0%)</td>
<td>227</td>
<td>975 (95.5%)</td>
</tr>
</tbody>
</table>

baselines (LLaMA2-Code, Mistral-Code and CodeQwen1.5). The evaluation results show that COT-SPARQL significantly outperforms the LLaMA2-Code model significantly on all datasets. For instance, on the LC-QuAD 2.0 dataset, (COT-SPARQL\(_\text{ent+rel}\)) , which incorporates both entities and relations, generates 4640 (96%) Valid queries and only 227 invalid queries. In comparison, the LLaMA2-Code model generates 1216 (25.3%) Valid queries and 3651 (74.7%) Invalid queries. These findings suggest that prompts enriched with In-context Learning and few-shot examples significantly enhance the ability of LLMs to generate valid and correct SPARQL queries than relying only on their pre-trained knowledge. Furthermore, the results show that our approach with both entities and relations (COT-SPARQL\(_\text{ent+rel}\)) consistently achieves the highest performance, compared to the variant with only entities (COT-SPARQL\(_\text{ent}\)).

5.3. Executing SPARQL queries in question answering (Pilot Study)

To address RQ3, which investigates the performance of our approach in question answering task, we used the GERBIL benchmark framework [43] to execute the generated SPARQL queries. Our goal is to evaluate the effectiveness of our approach in an end-to-end setting, where a natural language question is given as an input, converted into a SPARQL query and then executed to retrieve answers. In particular, we performed a pilot study on the QALD-10 dataset [39], the most recent benchmark dataset for questions answering over linked data. Furthermore, we compared the performance of our approach with the state-of-the-art baselines from GERBIL framework, namely:

- Kovriguina et al. [32] employed the GPT-3 model with one-shot example to generate a SPARQL query.
- Borroto and Ricca [44] combined neural machine translation with named entity recognition to convert natural language questions into SPARQL queries.
- Guo et al. [45] developed a system that classifies questions and generates SPARQL queries using templates and a knowledge base.
- Steinmetz et al. [46] introduced a pattern-based method to transform natural language into SPARQL queries by matching patterns and fill variables with relevant information from the question.
- Baramia et al. [47] presented a ranking method to optimize question answering over knowledge graph, focusing on ranking items to construct SPARQL queries.

As shown in Table 4, COT-SPARQL\(_\text{ent+rel}\) outperforms the state-of-the-art baseline [44] by achieving the higher macro F1-QALD score of 63.87. The full results from GERBIL.
Table 4. A pilot study of question answering task over the QALD-10 dataset. For the system of Kovriguina et al. [32], we were unable to find results on the QALD-10 leaderboard, therefore we obtained results from [32] (RQ3). Best results are in bold.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
<th>F1-QALD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kovriguina et al. [32]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.29*</td>
</tr>
<tr>
<td>Borroto and Ricca [44]</td>
<td>0.4538</td>
<td>0.4574</td>
<td>0.4538</td>
<td>0.5947</td>
</tr>
<tr>
<td>Guo et al. [45]</td>
<td>0.5068</td>
<td>0.5238</td>
<td>0.5070</td>
<td>0.5776</td>
</tr>
<tr>
<td>Steinmetz et al. [46]</td>
<td>0.3206</td>
<td>0.3312</td>
<td>0.3215</td>
<td>0.4909</td>
</tr>
<tr>
<td>Baramiia et al. [47]</td>
<td>0.4289</td>
<td>0.4272</td>
<td>0.4277</td>
<td>0.4281</td>
</tr>
<tr>
<td>COT-SPARQL-(ent+rel)</td>
<td>0.4944</td>
<td>0.5072</td>
<td>0.4978</td>
<td><strong>0.6387</strong></td>
</tr>
</tbody>
</table>

Figure 4. The detail results of our approach in GERBIL benchmark framework.

16 are also shown in Figure 4. Overall, the evaluation results (shown in the top row of Figure 4) indicates that our queries are more reliable in retrieving correct answers from knowledge graphs. Furthermore, the second row of Table 4 represents GERBIL’s [43] sub-experiment called Concept to Knowledge Base (C2KB), which identifies all resources that are relevant for the given question. In particular, GERBIL calculates the measures precision, recall and F-measure based on the comparison of the expected resource URIs and the URIs returned by the QA system. The third row of Table 4 shows the Properties to Knowledge Base (P2KB) sub-experiment, where GERBIL identifies all properties that are relevant for the given question. The last row of Table 4 represents Relation to Knowledge Base (RE2KB) sub-experiment, which focuses on the triples that have to be extracted from the question and are needed to generate the SPARQL query for retrieving correct answers. The full evaluation results can be accessed via the public KGQA leaderboard.17

16Experiment link at GERBIL framework: https://gerbil-qa.aksw.org/gerbil/experiment?id=20240514002
6. Conclusion and Future Work

This paper presents a novel approach for SPARQL generation, leveraging In-context learning and Chain-of-Thoughts prompt in large language models to generate high-quality SPARQL queries from natural language. Specifically, we incorporate additional context information from the input question, including entities and relations, into the Chain-of-Thought prompt. Furthermore, we include a semantically similar one-shot example within the prompt to facilitate generating precise SPARQL queries. In contrast to existing methods relying on pre-defined templates or fixed rules, our approach is capable of adapting to generate diverse SPARQL syntax tailored to a target knowledge graph. To assess the effectiveness of our approach, we conducted experiments on various benchmark datasets. The results demonstrate that our method outperforms state-of-the-art methods in terms of both accuracy and the validity of generated SPARQL queries. In our future research, we plan to investigate fine-tuning large language models (e.g., LlaMA2-Code) on multitask learning for both SPARQL generation and question answering over knowledge graphs.

Acknowledgement

This work has been supported by the German Federal Ministry of Education and Research (BMBF) within the projects, COLIDE (grant no 01I521005D), KIAM (grant no 02L19C115), the European Union’s Horizon Europe research and innovation programme (grant No 101070305), the Ministry for Economic Affairs, Innovation, Digitalisation and Energy of North Rhine-Westphalia (MWIDE NRW) within the project Climate bOWL (grant no 005-2111-0020), and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation): TRR 318/1 2021 – 438445824.

References


