Question Answering over DBpedia with Fine-tuned Autoregressive Models

Tommaso Soru[1,2], Saurav Joshi[1,3], Sanju Tiwari[1,4], Mehrzad Shahinmoghadam[1] and Anand Panchbhai[1]

[1]*DBpedia Association; Leipzig, Germany*
[2]*Liber AI Ltd; London, United Kingdom*
[3]*University of Southern California; Information Sciences Institute; Marina del Rey, CA, USA*
[4]*Sharda University; Greater Noida, India*

Abstract

This paper presents a novel approach to question answering over DBpedia by fine-tuning autoregressive code generation models to translate natural language questions into SPARQL queries. The authors introduce the NSpM Dataset, a large-scale dataset of 7.7 million question-SPARQL pairs covering DBpedia's ontology. We fine-tuned three open-source code generation LLMs – CodeGen-350M, StarCoder-1B, and CodeLlama-7B – on a curated subset of this dataset. Despite limited training time, StarCoder-1B achieved the best performance across evaluation metrics. Results from the 1st Text2SPARQL Challenge at ESWC 2025 reveal that while models partially learn SPARQL syntax, they struggle with query semantics and ranking quality.

Keywords

Semantic Web, SPARQL, semantic parsing, question answering, neural networks, large language models

1. Introduction

In recent years, large knowledge graphs (such as DBpedia [1], Wikidata [2], and Freebase [3]) have grown to contain more than billions of semantic triples forming massive repositories of structured knowledge. To democratize this knowledge, Question Answering (QA) systems have been developed to let users retrieve information from these knowledge bases using natural language questions instead of requiring them to write complex queries in formal languages like SPARQL [4, 5, 6]. However, translating natural language questions onto the rich ontology of a large graph and generating an executable SPARQL statement remains highly challenging, because a system must resolve lexical ambiguity, link surface terms to the correct entities and relations, preserve graph-structural constraints, and generate syntactically valid queries that support multi-hop, and compositional reasoning [7, 8, 9, 10, 11].

Early computational research on Text-to-SPARQL has predominantly relied on rule and template based systems that align parsed questions with fixed SPARQL skeletons; representative examples are the Template-Based Question Answering over Linked Data (TBSL) framework [12] and the keyword-to-template generator described in [13], which require extensive manual engineering and fail to generalize to paraphrases, unseen ontologies, and multi-hop queries. Semi-automatic approaches—notably the skeleton-grammar with neural-ranking model SPARQA [14] and the classifier-guided template-selection method proposed in [15]—learn and rank patterns from data, reducing authoring effort yet still inherit rigidity from their finite template data stores and remain brittle on out-of-distribution question forms.

In this paper, we contribute to the Text-to-SPARQL research through two primary advancements. First, we introduce the NSpM dataset, a massive, automatically-generated collection comprising 7.7 million natural language question–SPARQL pairs that span all ontology classes within DBpedia. This dataset is carefully assembled using an extensive systematic template discovery and generation framework, augmented by question paraphrasing and enhanced with commonsense knowledge validation to ensure

First International TEXT2SPARQL Challenge, Co-Located with Text2KG at ESWC25, June 01, 2025, Portorož, Slovenia.

tom@tommaso-soru.it (T. Soru)
http://tommaso-soru.it (T. Soru)
0000-0002-1276-2366 (T. Soru)

logical coherence and coverage of diverse and compositional query patterns. Second, we fine-tune three specialized open-source code large language models—specifically CodeGen-2.5 (350M), StarCoder (1B), and CodeLlama (7B)—on our dataset, enhancing their ability to generate structurally accurate and ontology-aware SPARQL queries. By leveraging pretrained models explicitly designed for code generation, our fine-tuned models effectively internalize query syntax and semantics, significantly improving practical applicability in querying large-scale knowledge graphs. Both the NSpM dataset and our fine-tuned models are released as open artifacts to further research and innovation in neural query generation over structured knowledge bases.

The paper is structured as follows. In Section 2, we introduce the works relevant to this research field. Then in Section 3, we outline the approach from idea to execution. Section 4 discusses the pre- and post-submission evaluations, which we elaborate on in Section 5. Finally, we conclude.

2. Related Work

At the time of writing, many recent works have applied neural sequence models to the Text-to-SPARQL task. For example, a pointer-network model with relation-aware attention was introduced in [16], and large pre-trained language models like T5, BART, or pointer-generator networks were shown to effectively arrange entities and relations into SPARQL queries [17]. In a similar vein, SGPT (stacked Transformer encoders + GPT-2) was proposed in [6], but it often fails to capture complex multi-hop graph relations, leading to mistakes in the generated triples. Prompt-based methods (e.g., one-shot SPARQL generation by injecting relevant KG subgraphs into the model's context window) have been explored in [18], yet they still exhibit systematic errors such as the "triple-flip" (swapping subject and object positions in a triple). A Triplet-Order-Sensitive pre-training for T5 was introduced in [11] to correct subject/object ordering. However, all of these approaches rely on costly fine-tuning and specialized decoding schemes, making them resource-intensive and potentially brittle across different knowledge graphs [16, 18, 11].

Recent work on neural Text-to-SPARQL, particularly over the DBpedia knowledge graph, has produced several end-to-end models evaluated extensively on DBpedia-specific benchmarks. For example, an encoder–decoder model that integrates KG structure directly into decoding architecture achieving state-of-the-art F1 on LC-QuAD 1.0 [19]. A hybrid multi-head convolutional encoder combining CNN and self-attention reported strong BLEU and F1 scores on QALD-9 and LC-QuAD 1.0 [20]. A universal KG-QA system that casts question understanding as sequence generation achieved competitive performance on LC-QuAD 1.0 and QALD-9 [21]. A two-stage ontology-guided prompting framework, which first predicts a SPARQL skeleton and then completes it with knowledge graph specific information, achieved 79.1% F1 on LC-QuAD 1.0 [22]. Earlier work also proposed a silhouette-based two-step architecture combining coarse query generation and graph search, showing substantial gains on LC-QuAD 1.0 [23].

Large-scale QA benchmarks have been introduced to support training and evaluation. For example, the LC-QuAD 2.0 dataset contains on the order of 30,000 complex natural-language questions paired with SPARQL queries over DBpedia (and Wikidata) [9]. Its size, diversity, and dual-KG support make it a unique resource for both academic research and industrial applications in KG-based dialogue, interpretable AI and semantic search. The QALD challenge series provides smaller, multilingual benchmarks; QALD-9-Plus extends QALD-9 with high-quality DBpedia question translations into eight languages in KGQA and 5 underrepresented languages, i.e. Ukrainian, Armenian, Lithuanian, Bashkir, Belarusian. [24]. More recently, the DBpedia Neural QA (DBNQA) corpus was released, offering approximately 894,499 English question–SPARQL pairs generated via templating for DBpedia [25]. These resources cover a wide variety of query patterns and have become standard datasets for benchmarking DBpedia QA methods.

3. Approach

The central objective of our project was to evaluate the capability of pretrained autoregressive code generation models to perform text-to-SPARQL translation. Our initial plan envisioned leveraging a general-purpose GPT-style language model; however, upon review, the recent proliferation of open-source models tailored to code generation prompted a pivot in strategy. These models, often trained on rich corpora of source code, offer potentially advantageous inductive biases for structured language tasks like SPARQL query generation.

3.1. Model Selection

We performed a comparative review of several publicly available autoregressive models, focusing on:

- Model size and accessibility
- Training data composition and license constraints
- Prior performance on code-related tasks

Ultimately, we selected the following three models for fine-tuning:

- A Salesforce *CodeGen* model [26] with 350 million parameters
- A BigCode *StarCoder* model [27] with 1 billion parameters
- A Meta *Code Llama* model [28] with 7 billion parameters

These models represent a diverse trade-off between model size, inference cost, and representational capacity. We expected CodeLlama-7b in particular to provide a valuable upper-bound comparison due to its substantial scale and recent development.

3.2. Dataset creation

As an initial step, we created SPARQL query skeleton representations for core ontology classes within the DBpedia ontology, such as Person and Work and grouped them into five linguistic families: subordinate, conjunctive/disjunctive, comparative, superlative, and numeric. A T5-base encoder–decoder model [29], fine-tuned for question generation, turned each representation into a fluent natural-language template while preserving its structure. This process yielded examples such as "Where is <A> born?" (subordinate), "Who is the parent of <A> and <B>?" (conjunctive), "Has <A> had a child?" (comparative), "What is the highest elevation of <A>?" (superlative), and "How many children did <A> have?" (numeric). Collectively, these templates cover single-entity attributes, multi-entity relations, boolean conditions, minimum-maximum value queries, and explicit counting patterns.

Subsequent steps focused on quality assurance to ensure we created a high-quality dataset for the model to learn from and generalize. First, we computed cosine similarity to discard paraphrase pairs falling below 0.90 threshold, thereby eliminating semantically divergent rewrites. Next, a commonsense classifier removed logically implausible questions, and a duplicate-detection model suppressed near-identical variants. Placeholders were then instantiated with actual DBpedia URIs—sampling additional entities for multi-entity questions and inserting values that guarantee coherent boolean or aggregate answers. After validation on Person and Work, the identical pipeline was applied to all the ontologies present in DBpedia such as Organisation, Place, and Event, yielding the NSpM corpus of 7.7 million natural-language question and SPARQL query pairs that are used in our downstream Text-to-SPARQL experiments.

3.3. Research Exploration

3.3.1. Model architectures

During the start of our research journey, we focused extensively on understanding the internal architecture and design goals of the CodeGen family, particularly the 350M and 7B multi-language variants.

These models, built for translating natural language into executable code, offered a promising avenue for our SPARQL translation task. We also examined the NSQL models – fine-tuned descendants of CodeGen trained specifically for SQL query generation – which provided useful architectural and prompt formatting insights transferable to SPARQL [30].

A significant insight emerged from analyzing the "the-stack" dataset, which underpins the CodeGen models. Despite its impressive breadth, only $\sim$24,000 SPARQL-related samples were identified, compared to millions for general code. This data scarcity emphasized the need for synthetic data augmentation to supplement fine-tuning. Additionally, we studied NSQL's technique of schema-augmented prompt formatting and considered applying ontology-based embeddings from DBpedia to improve prompt quality. On the engineering side, inference time for the 7B model exceeded 80 seconds on CPU, necessitating deployment on dual RTX 3090 GPUs, which cut it to under 6 seconds.

3.3.2. Prompt engineering

As we moved onto the next stage, the focus shifted to enhancing the formatting of prompts. We refined the placement of task instructions and demarcated clear input-output separators. The inclusion of schema metadata and prefixes was explored as a means to guide the model toward more accurate generation of SPARQL queries. These changes were based on the hypothesis that clearer, context-enriched prompts would improve the model's understanding of the translation task.

We also expanded our few-shot experimentation by varying domains and intent types. This allowed us to test the generalization capabilities of the models when exposed to structurally and semantically diverse examples. In parallel, we began evaluating model outputs using the QALD dataset to establish a secondary benchmark beyond the Text2SPARQL dataset. This benchmarking effort is expected to shed light on model adaptability across multiple query contexts.

3.3.3. Issues with instruct models

We shifted our attention to models with instruction-following capabilities. The instruct-tuned variant of CodeGen, namely `codegen25-7b-instruct`, was selected for experimentation under the hypothesis that such models could better translate detailed user queries into structured outputs like SPARQL. Preliminary trials revealed promising but mixed results – while the model demonstrated aptitude in understanding instructions, it struggled with output length control and formatting.

We addressed these issues through two means: first, by introducing the `<eom>` (End of Message) token into the prompt, followed by a post-processing step to truncate outputs accordingly; and second, by inspecting and adjusting the prompt-to-response formatting pipeline. This highlighted a broader challenge – the pretraining corpus likely influenced the model to favor line-by-line SPARQL formatting regardless of prompt style, revealing a deeper inductive bias in structure.

3.3.4. Model fine-tuning

We turned toward constructing a dedicated prompt-tuning dataset to support more targeted fine-tuning. We chose the `StarCoder-1b` checkpoint as our initial model due to its manageable size and architecture compatibility. The first step involved designing and implementing prompt formatting modules that adhered to both the structural norms of code-based language models and the functional requirements of DBpedia queries.

In parallel, we began harvesting and transforming data from the NSpM dataset, which offers a substantial repository of over 7 million (NL question, SPARQL query) pairs. We extracted and refined a subset of 100,000 samples for initial experiments, ensuring compatibility with model expectations. The refinement process involved standardizing schema references, enforcing a clean prompt-completion structure, and enriching each prompt with auxiliary context to improve model comprehension.

We initiated the actual fine-tuning process on StarCoder-1B using our customized dataset and a modified version of the official fine-tuning script. This required adjustments to handle the structure of our NSpM-derived samples and integrate them with the BigCode and HuggingFace training pipelines.

Despite encountering expected hardware constraints, we successfully launched training and saved intermediate checkpoints at steps 1,000 and 2,000 to facilitate upcoming evaluation rounds, where a step is the fine-tuning of the model's weights using a batch of question-query pairs. Early runtime profiling also helped us estimate the full duration of training, prompting discussions around possible trade-offs in data size and training duration.

All fine-tuned models are available for download on the HuggingFace platform and via their transformers library.[1]

3.4. Training Setup

Fine-tuning was conducted using the HuggingFace Transformers library. Key parameters included:

- Learning rate: 5e-5
- Optimizer: AdamW
- Batch size: 8
- Max sequence length: 512

Due to hardware resources and availability limitations, we were only able to complete approximately 20% of the originally scheduled training steps. This affected model convergence, especially for the larger models, which typically require longer training horizons to fully adapt. CodeLlama-7b, in particular, could not be fine-tuned in full also due to its high-memory footprint.

In our setup, fine-tuning followed the standard causal language modeling objective, wherein the model learns to predict the next token in the SPARQL output given the preceding tokens from the concatenated prompt–completion pair. All models were initialized from their publicly released checkpoints, tokenized using their native tokenizers, and trained with gradient accumulation to accommodate larger effective batch sizes on the available GPUs.

4. Evaluation

The development of this project started around two years before the submission to the Text2SPARQL challenge. Due to this fact, we proceeded to our own evaluation ahead of that carried out by the challenge organizers.

4.1. Preliminary evaluation

Before the official challenge dataset was made available, we conducted internal evaluations using automatic metrics. Specifically, we employed the BLEU score to measure the lexical overlap between generated SPARQL queries and their ground truth counterparts. While BLEU is commonly used in natural language generation tasks, its application here provided a coarse approximation of semantic similarity. Although it fails to capture query correctness in logical terms, it was useful in identifying egregious deviations and monitoring general output fidelity during model development.

4.2. Challenge evaluation

Besides the usual information retrieval metrics such as precision, recall, and F1 score, the performance was computed also for the Normalized Discounted Cumulative Gain at K (NDCG). Precision, recall, and F1-score are defined as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

[1] https://huggingface.co/dbpedia/models

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

where:

- TP: True Positives.
- FP: False Positives.
- FN: False Negatives.

Discounted Cumulative Gain is defined as follows:

$$\text{DCG}_k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)} \tag{4}$$

$$\text{NDCG}_k = \frac{\text{DCG}_k}{\text{IDCG}_k} \tag{5}$$

where:

- rel_i: Relevance score of the item at position i.
- DCG_k: Discounted Cumulative Gain at rank k. It measures ranking quality by assigning higher importance to relevant items appearing earlier in the list.
- $\log_2(i+1)$: Logarithmic discount factor, which reduces the contribution of lower-ranked items.
- IDCG_k: Ideal DCG at rank k, i.e., the maximum possible DCG obtainable with a perfect ranking of the top k results.
- NDCG_k: Normalized DCG, which ensures the score lies between 0 and 1. A value of 1.0 indicates a perfect ranking.

Metric	**CodeGen-350m-ft**	**StarCoder-1b-ft**	**CodeLlama-7b-ft**
Set Recall	0.055	0.100	0.053
Set Precision	0.055	0.097	0.049
Set F1 Score	0.055	0.098	0.051
NDCG	0.000	0.000	0.000
Set F1 + NDCG	0.055	0.098	0.050
Set F1 Score (EN)	0.060	0.108	0.066
Set Precision (EN)	0.060	0.107	0.064
Set Recall (EN)	0.060	0.110	0.068
NDCG (EN)	0.000	0.000	0.000
Set F1 + NDCG (EN)	0.060	0.108	0.066
Set F1 Score (ES)	0.050	0.088	0.036
Set Precision (ES)	0.050	0.087	0.034
Set Recall (ES)	0.050	0.090	0.038
NDCG (ES)	0.000	0.000	0.000
Set F1 + NDCG (ES)	0.050	0.088	0.036

Table 1
Evaluation metrics across the three fine-tuned code generation models.

The results obtained by our three fine-tuned models on the held-out challenge test set can be seen in Table 1. The table presents a comparative evaluation of three fine-tuned code generation models across various retrieval metrics. *StarCoder-1b-ft* consistently outperforms the other configurations, achieving the highest scores in set-based metrics such as Precision, Recall, and F1, as well as in their language-specific (EN/ES) variants. While *CodeGen-350m-ft* and *CodeLlama-7b-ft* exhibit similar performance in most metrics, the latter lags behind in Spanish-specific evaluations. Notably, all configurations yield an NDCG of 0, suggesting poor ranking quality despite acceptable set-based retrieval performance. Within the Text2SPARQL challenge leaderboard, our models ranked between the 9th and 12th place.

5. Discussion

Performance was consistent across both languages. Despite promising intermediate results during training, the models underperformed in generalization – likely due to incomplete fine-tuning.

5.1. Qualitative Error Analysis

The most frequent categories of failure were noted, as can be seen in the examples in this subsection:

- Incorrect variable bindings or predicate usage
- Hallucinated namespaces or entities
- Syntax errors in SPARQL structure

Notably, the StarCoder-based model showed a higher rate of syntactically valid but semantically incorrect queries, indicating partial learning of SPARQL structure.

```
Prompt:     # Is the emerald number of parking spaces of claridge icon ?

Actual:     ask where{dbr:Claridge_Icon dbo:numberOfParkingSpaces
            dbr:The_Emerald_(building) }
Processed:  ask where{dbr:Claridge_Icon dbo:numberOfParkingSpaces
            dbr:Emerald }
----------------------------------------------------------------------
Prompt:     # Did the george houston house have the NRHP reference number ?

Actual:     ask where{dbr:George_Houston_House dbo:nrhpReferenceNumber ?x }
Processed:  ask where{dbr:The_George_Houston_House
            dbo:nrhpReferenceNumber ?x }
----------------------------------------------------------------------
Prompt:     # How should I know the ingredient names for mache ?

Actual:     select ?x where{dbr:Mache_(food) dbo:ingredientName ?x }
Processed:  select ?x where{dbr:Mache dbo:ingredientName ?x }
----------------------------------------------------------------------
Prompt:     # Which was the year of the first ascent of laila peak and
            michael peak ?

Actual:     select ?x where{dbr:Laila_Peak_(Haramosh_Valley)
            dbo:firstAscentYear ?x . dbr:Michael_Peak dbo:firstAscentYear
            ?x }
Processed:  select ?x where{dbr:Michael_Peak dbo:firstAscentYear ?x.
            dbr:Laila_Peak dbo:firstAscentYear ?x }
----------------------------------------------------------------------
Prompt:     # Is soviet union date of liberation of ravensbrück
            concentration camp ?

Actual:     ask where{dbr:Ravensbrück_concentration_camp dbo:liberationDate
            Soviet Union, 30 April 1945 }
Processed:  ask where{dbr:Ravensbrück_Concentration_Camp
            dbo:dateOfLiberation dbr:Soviet_Union }
----------------------------------------------------------------------
Prompt:     # What is the using country of boliviano ?
```

```
Actual:     select ?x where{dbr:Boliviano_(1864-1963) dbo:usingCountry ?x }
Processed:  select ?x where{dbr:Boliviano dbo:usingCountry ?x }
-------------------------------------------------------------------------------
Prompt:     # Does bishop’s college have a Campus Type ?

Actual:     ask where{dbr:Bishop’s_College,_Colombo dbo:campusType ?x }
Processed:  ask where{dbr:Bishop’s_College dbo:campusType ?x }
-------------------------------------------------------------------------------
Prompt:     # Is beer actually lone tree brewery product ?

Actual:     ask where{dbr:Lone_Tree_Brewery dbo:product dbr:Beer }
Processed:  ask where{dbr:Lone_Tree_Brewery dbo:product ?x }
```

5.2. Reflections

The outcomes of this project reflect both the potential and limitations of autoregressive code models for structured query generation. While our selection of models was theoretically motivated, practical constraints significantly impacted performance.

One of the promising signs was that even with limited fine-tuning, models adapted partially to the SPARQL grammar. This suggests that autoregressive code models, particularly those with instruction tuning, can generalize to formal languages given suitable conditioning.

Model performance across English and Spanish was roughly equivalent. Since none of the models were explicitly multilingual, this implies that alignment was primarily pattern-based rather than grounded in semantic understanding. Future work could incorporate multilingual pretraining or contrastive SPARQL alignment strategies.

5.3. Future improvements

- Secure longer training cycles for larger models to enable full convergence
- Explore SPARQL-specific prompting patterns to enforce output structure
- Integrate syntactic constraints or symbolic verification at decoding time
- Investigate adapter-based multilingual extensions

Securing longer training cycles for larger models remains a critical step toward achieving full convergence. Limited compute budgets often lead to premature stopping, which hinders the model’s ability to internalize more nuanced patterns, particularly in complex language generation tasks. Future efforts should allocate sufficient computational resources to support multi-epoch training and explore techniques such as curriculum learning and checkpoint averaging to accelerate and stabilize convergence.

Another promising direction involves exploring SPARQL-specific prompting patterns to enforce output structure more reliably. Prompt engineering tailored to the syntactic and semantic characteristics of SPARQL could significantly improve model alignment with query constraints. Strategies such as providing canonical templates, embedding intermediate representations, or framing generation as structured completion may help reduce syntactic errors and hallucinations.

In addition, incorporating syntactic constraints or symbolic verification during decoding could further enhance output validity. By integrating rule-based filters, grammar checkers, or programmatic validators at inference time, the system can eliminate structurally invalid outputs before finalization. This hybrid approach leverages both the generative strengths of autoregressive models and the precision of symbolic systems, promoting trustworthiness in executable query generation.

Lastly, investigating adapter-based multilingual extensions may allow efficient cross-lingual generalization without retraining entire models. Adapters can be used to insert lightweight, language-specific modules within a shared architecture, enabling parameter-efficient fine-tuning. This setup is particularly advantageous in resource-constrained settings or when scaling support to additional languages with limited data.

6. Conclusion

We introduced a method for knowledge graph question answering based on fine-tuning large language models for code generation. Moreover, we release a novel dataset of 7.7 million pairs of questions and queries against the DBpedia knowledge base. Despite not performing splendidly, our approach was potentially limited by the available resources. First, results could be further improved by utilizing 100% of the data during fine-tuning; second, by securing longer training cycles for larger models to enable full convergence. Exploring SPARQL-specific prompting patterns could help enforce output structure. Lastly, integrating syntactic constraints or symbolic verification at decoding time may make the output more robust.

Acknowledgments

This work was carried out over Summer 2023 during the Google Summer of Code programme at DBpedia. The research belongs to project *A Neural Question Answering Model for DBpedia*, which spanned from 2018 to 2023. We thank all mentors and contributors who took part in it.

Declaration on Generative AI

In the writing of this manuscript, we employed generative artificial intelligence for sentence polishing, proofreading, and rephrasing. The authors take full accountability for the veracity of its content.

References

[1] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, C. Bizer, Dbpedia – a large-scale, multilingual knowledge base extracted from wikipedia, Semantic Web 6 (2015) 167–195. URL: https://journals.sagepub.com/doi/abs/10.3233/SW-140134. doi:10.3233/SW-140134. arXiv:https://journals.sagepub.com/doi/pdf/10.3233/SW-140134.

[2] D. Vrandečić, M. Krötzsch, Wikidata: A free collaborative knowledgebase, Communications of the ACM 57 (2014) 78–85. doi:10.1145/2629489.

[3] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, J. Taylor, Freebase: a collaboratively created graph database for structuring human knowledge, in: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008, pp. 1247–1250.

[4] T. Soru, E. Marx, D. Moussallem, G. Publio, A. Valdestilhas, D. Esteves, C. B. Neto, Sparql as a foreign language, in: J. D. Fernández, S. Hellmann (Eds.), Proceedings of the Posters and Demos Track of the 13th International Conference on Semantic Systems - SEMANTiCS2017, number 2044 in CEUR Workshop Proceedings, Aachen, 2017. URL: http://ceur-ws.org/Vol-2044/paper14.

[5] T. Soru, E. Marx, A. Valdestilhas, D. Esteves, D. Moussallem, G. Publio, Neural machine translation for query construction and composition, 2nd Workshop on Neural Abstract Machines and Program Induction (NAMPI v2) at ICML 2018 (2018).

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

[7] S. Cao, J. Shi, L. Pan, L. Nie, Y. Xiang, L. Hou, J. Li, B. He, H. Zhang, KQA pro: A dataset with explicit compositional programs for complex question answering over knowledge base, in: S. Muresan, P. Nakov, A. Villavicencio (Eds.), Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational

Linguistics, Dublin, Ireland, 2022, pp. 6101–6119. URL: https://aclanthology.org/2022.acl-long.422/. doi:10.18653/v1/2022.acl-long.422.

[8] S. Liang, K. Stockinger, T. M. de Farias, M. Anisimova, M. Gil, Querying knowledge graphs in natural language, J. Big Data 8 (2021) 3.

[9] M. Dubey, D. Banerjee, A. Abdelkawi, J. Lehmann, Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia, in: The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019, Proceedings, Part II 18, Springer, 2019, pp. 69–78.

[10] M. Mountantonakis, Y. Tzitzikas, Generating sparql queries over cidoc-crm using a two-stage ontology path patterns method in llm prompts, J. Comput. Cult. Herit. 18 (2025). URL: https://doi.org/10.1145/3708326. doi:10.1145/3708326.

[11] J. Qi, C. Su, Z. Guo, L. Wu, Z. Shen, L. Fu, X. Wang, C. Zhou, Enhancing sparql query generation for knowledge base question answering systems by learning to correct triplets, Applied Sciences 14 (2024). URL: https://www.mdpi.com/2076-3417/14/4/1521. doi:10.3390/app14041521.

[12] C. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, P. Cimiano, Template-based question answering over rdf data, in: Proceedings of the 21st International Conference on World Wide Web, WWW '12, Association for Computing Machinery, New York, NY, USA, 2012, p. 639–648. URL: https://doi.org/10.1145/2187836.2187923. doi:10.1145/2187836.2187923.

[13] S. Shekarpour, S. Auer, A.-C. Ngonga Ngomo, D. Gerber, S. Hellmann, C. Stadler, Generating sparql queries using templates, volume 11, 2013. doi:10.3233/WIA-130275.

[14] Y. Sun, L. Zhang, G. Cheng, Y. Qu, Sparqa: Skeleton-based semantic parsing for complex questions over knowledge bases, Proceedings of the AAAI Conference on Artificial Intelligence 34 (2020) 8952–8959. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6426. doi:10.1609/aaai.v34i05.6426.

[15] A. Formica, I. Mele, F. Taglino, A template-based approach for question answering over knowledge bases, Knowl. Inf. Syst. 66 (2023) 453–479. URL: https://doi.org/10.1007/s10115-023-01966-8. doi:10.1007/s10115-023-01966-8.

[16] J. Zou, M. Yang, L. Zhang, Y. Xu, Q. Pan, F. Jiang, R. Qin, S. Wang, Y. He, S. Huang, et al., A chinese multi-type complex questions answering dataset over wikidata, arXiv preprint arXiv:2111.06086 (2021).

[17] D. Banerjee, P. A. Nair, J. N. Kaur, R. Usbeck, C. Biemann, Modern baselines for sparql semantic parsing, in: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2022, pp. 2260–2265.

[18] L. Kovriguina, R. Teucher, D. Radyush, D. Mouromtsev, Sparqlgen: One-shot prompt-based approach for sparql query generation., in: SEMANTiCS (Posters & Demos), 2023.

[19] J. Lee, H. Shin, Sparkle: Enhancing sparql generation with direct kg integration in decoding, Expert Systems with Applications (2025) 128263.

[20] Y.-H. Chen, E. J.-L. Lu, K.-H. Cheng, Integrating multi-head convolutional encoders with cross-attention for improved sparql query translation, arXiv preprint arXiv:2408.13432 (2024).

[21] R. Omar, I. Dhall, P. Kalnis, E. Mansour, A universal question-answering platform for knowledge graphs, Proceedings of the ACM on Management of Data 1 (2023) 1–25.

[22] L. Jiang, J. Huang, C. Möller, R. Usbeck, Ontology-guided, hybrid prompt learning for generalization in knowledge graph question answering, arXiv preprint arXiv:2502.03992 (2025).

[23] S. Purkayastha, S. Dana, D. Garg, D. Khandelwal, G. S. Bhargav, A deep neural approach to kgqa via sparql silhouette generation, in: 2022 International Joint Conference on Neural Networks (IJCNN), IEEE, 2022, pp. 1–8.

[24] A. Perevalov, D. Diefenbach, R. Usbeck, A. Both, Qald-9-plus: A multilingual dataset for question answering over dbpedia and wikidata translated by native speakers, in: 2022 IEEE 16th International Conference on Semantic Computing (ICSC), IEEE, 2022, pp. 229–234.

[25] A.-K. Hartmann, E. Marx, T. Soru, Generating a large dataset for neural question answering over the dbpedia knowledge base, in: Workshop on Linked Data Management, co-located with the W3C WEBBR, volume 2018, 2018.

[26] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, C. Xiong, Codegen: An open large language model for code with multi-turn program synthesis, arXiv preprint arXiv:2203.13474 (2022).

[27] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, et al., Starcoder: may the source be with you!, arXiv preprint arXiv:2305.06161 (2023).

[28] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, et al., Code llama: Open foundation models for code, arXiv preprint arXiv:2308.12950 (2023).

[29] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, Journal of machine learning research 21 (2020) 1–67.

[30] A. Chopra, R. Azam, Enhancing natural language query to sql query generation through classification-based table selection, in: L. Iliadis, I. Maglogiannis, A. Papaleonidas, E. Pimenidis, C. Jayne (Eds.), Engineering Applications of Neural Networks, Springer Nature Switzerland, Cham, 2024, pp. 152–165.