Schema-Constrained Grammar-Guided Generation of GQL Queries from Natural Language

André Melo[1,2,*,†], Jeff Z. Pan[2,†]

[1]2 Semple St., EH3 8BL Edinburgh, United Kingdom

Abstract

Graph databases provide a robust foundation for storing and querying knowledge graphs, making them well-suited for powering intelligent personal assistants. Translating natural language into graph query languages (NL2GQL) enables these assistants to answer user queries by leveraging personal data stored as graphs. To preserve user privacy, on-device processing is preferred, but it comes with strict computational and memory constraints. A major challenge in this setting is ensuring that generated queries comply with user-specific, often custom, graph schemata. In this paper, we explore the business value and challenges of NL2GQL translation in a privacy-sensitive, resource-constrained setting, and propose grammar-guided generation with incorporated schema constraints as a potential solution to address these challenges.

Keywords

grammar guided generation, schema constrained generation, knowledge graphs, question answering

1. Introduction

Large Language Models (LLMs) are powerful tools offering advanced capabilities in understanding and generating natural language, which makes them ideal for personal assistant applications (e.g. Apple's Siri and Huawei's Celia). LLMs also have the ability to translate natural language (NL) into structured query languages, such as Graph Query Language (GQL) [1], enabling users to access and manipulate structured data without needing to understand complex query syntax. This allows on-device user information, which can be structured as a knowledge graph [1], to be used for answering complex personal questions, e.g., "What was the book my mom gave to my sister at her birthday last year?", which may require the connection of cross-application personal information from contacts, events, pictures, documents, etc. In such personal settings, on-device inference is essential to ensure privacy. This imposes strict resource requirements, limiting the size of the models that can be used. Whilst open weights models can translate NL queries with some success, their smaller versions (around 7B or less) struggle to generate correct and syntactically valid queries [2].

Additionally, the generated query also needs to comply with the graph schema. The usual approach for generating schema-compliant queries with LLMs is to add the relevant schema definition to the prompt context, which can substantially increase the context size and therefore the inference time. Alternatively, one could fine-tune a model to generate queries following a given graph schema. However, different users may have different schemata, thereby requiring a generic umbrella schema for all users, or fine-tuning the model for each user's schema. The model would also have to be updated every time there are changes to the schema. Adding a separate specialised model for the query translation is another option, but it is not memory efficient, which is a major issue in the on-device setting.

Grammar-constrained generation can enhance the LLM's graph query generation capabilities and improve the syntactic accuracy without requiring any fine-tuning or additional specialised models. Moreover, by creating custom grammar extensions, the schema information, or parts of it, can be

ISWC 2025 Companion Volume, November 2–6, 2025, Nara, Japan

*Corresponding author.

†These authors contributed equally.

andre.melo@huawei.com (A. Melo); jeff.pan@huawei.com (J. Z. Pan)

0000-0001-6158-7763 (A. Melo); 0000-0002-9779-2088 (J. Z. Pan)

[1]GQL is an upcoming standard language for property graph querying: https://www.gqlstandards.org/

integrated into the GQL grammar itself. This allows the LLM to generate schema-compliant queries off-the-bat and reduces or eliminates the need to provide the schema definition within the query translation prompt, thereby reducing compute and response time.

2. Related Work

There are several works NL2SQL queries using grammar, with the generation of the query's abstract syntax tree (AST) in a top-down manner [3, 4]. That means the model has to directly generate AST nodes, which requires special training data and vocabulary. These works later incorporate schema information to generate terminal nodes in the tree. Other methods rely on the LLM to generate valid queries, featuring refinement and correction prompting pipelines to progressively improve the query [5]. This is prohibitively expensive and may require several prompting steps with long contexts. It also does not guarantee syntax compliance and it is highly sensitive to the LLM's capabilities. Liang et al. fine-tunes LLMs for both the query language and the application schema [6], however, this is inherently expensive, requiring fine-tuning for each new schema, and requiring training data to be available.

More recently, SGU-SQL [7] reached state-of-the-art NL2SQL results with syntax-based decomposition to progressively generate SQL queries. It establishes connections between user queries and database schema and decomposes the complex generation task using syntax-based prompting. Zhout et al. [5] report low syntactic accuracy on NL2GQL translation with in-context schema declaration and no schema-constrained generation. Whilst Netz et al. [8] show that adding grammar-constrained generation to a model leads to substantial improvement in syntax accuracy, however, they report increase in runtime. To the best of our knowledge, there has been no works using grammar-guided generation, where the schema information is incorporated directly into the grammar itself.

3. Schema-Constrained Grammar-Guided Generation

Grammar-guided generation enables LLMs to generate output conforming to a given grammar. It has become increasingly popular, with specialized open-source systems such as transformersCFG [9] syncode [10] and outlines [11] enabling support of custom context-free-grammars (CFGs). XGrammar [12] improves decoding efficiency and inference runtime by dividing the vocabulary into context-independent tokens that can be pre-checked and context-dependent tokens that need to be interpreted during runtime. At the same time, popular LLM inference frameworks, such Llama.cpp, are also able to do rule-based constrained decoding, not only with regex and custom JSON schemata, but also with GBNF [2] support, which also enables use of custom CFGs.

3.1. Schema-Constrained Grammar Extension

The existence of grammar-constrained generation frameworks supporting custom CFG definitions provides invaluable opportunities for the query translation problem. A custom extended query language grammar including schema information is able to enforce both syntactic accuracy and schema compliance at decoding time. Moreover, a subset of the target GQL language can be used, in case the whole language syntax is not needed for a given use case. E.g., if the graph is meant to support question answering only, the parts of the GQL's language used for creation, update and deletion can be ignored.

In order to enable the schema-constrained grammar extension, the original query language grammar needs to be modified to include placeholders for the schema components. Those would then be expanded in order to include the input schema information, which can be done fully automatically. The difficulty and manual effort required is the inclusion of placeholders into the source query language grammar, which is done once per query language.

The placeholders in the example are for node type `${node}`, property `${prop}` and relation `${rel}`. Those are expanded by iterating over the schema components provided as input. Given the schema below, the

[2] https://github.com/ggml-org/llama.cpp/blob/master/grammars/README.md

schema-constrained grammar rules (right column) can be generated from the definition with placeholder (left). The rules names with more than one placeholder, e.g., ${rel}_${prop}, allowing the expressions of schema dependencies. In the example [3], plays_for_properties only allows points_scored and position as defined in the schema. This also allows property datatypes to be enforced according to the schema.

```
CREATE TAG ‘Player‘ (‘name‘ string NULL, ‘birthday‘ date NULL, ‘birthplace‘ string NULL, ‘retired‘ boolean NULL)
CREATE TAG ‘Team‘ (‘name‘ string NULL, ‘city‘ string NULL)
CREATE EDGE ‘PLAYS_FOR‘ (‘position‘ string NULL, points_scored int64 NULL)
CREATE EDGE ‘TEAM_MATE‘ (‘start_year‘ int64 NULL, ‘end_year‘ int64 NULL)
```

```
relationship_label_props: ${rel}_label_props
${rel}_label_props: ${rel}_label ${rel}_properties?
${rel}_properties: "{" ${rel}_property_list "}"
${rel}_property_list: ${rel}_property ("," ${rel}_property)*
${rel}_property: ${rel}_${prop}
${node}_${prop}: ${node}_${prop}_label ":" ${node}_${prop}_value
```

```
relationship_label_props: plays_for_label_props | team_mate_label_props
plays_for_label_props: "PLAYS_FOR" plays_for_properties?
plays_for_properties: ":" "{" plays_for_property_list "}"
plays_for_property_list: plays_for_property ("," plays_for_property)*
plays_for_property: plays_for_position | plays_for_points_scored
plays_for_position: "position" ":" STRING
plays_for_points_scored: "points_scored" ":" INT
team_mate_label_props: "TEAM_MATE" team_mate_properties?
```

The context-free grammar constraining the output does not ensure context-dependent compliance. Therefore, parts of the query that depend on context, e.g. variable references, could potentially violate the schema. Similarly, the literals in may require a post-processing step using the AST to ensure they are correctly grounded against the graph data.

3.2. Graph Schema Compliance and Data Grounding Post-processing

Syntactic correction allows the generation of the query's AST, which can be used in a post-processing step to support the validation and correction of variable references and query literals. For instance, given the schema definition, property assertion and GQL query below, the query's AST can be used to resolve the v2 variable ad identify it refers to a Team and that it is accessing the property birthplace. Since, according to the schema, birthplace is not a property for Team, the schema violation is detected.

```
CREATE TAG ‘Team‘ (‘name‘ string NULL, ‘city‘ string NULL)
(p:{name: “Michael Jordan”})-[PLAYS_FOR]->(t:{city: “Washington D.C”})
MATCH (v1:Player{name:”Jordan”})-[:PLAYS_FOR]->(v2:Team) WHERE v2.birthplace = “D.C.” RETURN v2.name
```

The AST can also be used for literal grounding by identifying the kind property being referenced in the literal expressions. E.g., in a string equality expression it is important to check whether the string used in the expression is not incorrect due to a typo or a match is missed due to usage of synonyms or abbreviations not contained in the graph. In the example above, "Jordan" can be checked against player names in the data graph to identify potential corrections, e.g. "Michael Jordan". This can substantially increase the query execution performance, as wrong query literals can lead to wrong results.

4. Conclusion and Future Research Directions

Integrating semantic knowledge-graph schemata into grammar has the potential to have a high impact on personal assistants, as it enables low-resource on-device natural language to graph query language translation. However, there are several challenges to overcome and questions to research. It has been shown that making the grammar too restrictive may hinder performance [13], therefore researching how to best incorporate the schema information into the grammar is important. Another interesting research topic is the incorporation of online parsing and observed context-dependent schema violations into the decoding process. Backtracking [14] strategies to deal with detected violations could be used to enable more comprehensive schema compliance. Finally, tools and best-practice guidelines for extending grammars with schema information need to be researched and developed.

[3]The example follows the Lark syntax and team_mate is not extended due to space restrictions

Declaration on Generative AI

Either:
The author(s) have not employed any Generative AI tools.

Or (by using the activity taxonomy in ceur-ws.org/genai-tax.html):
During the preparation of this work, the author(s) used X-GPT-4 and Gramby in order to: Grammar and spelling check. Further, the author(s) used X-AI-IMG for figures 3 and 4 in order to: Generate images. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] L. Liu, H. Du, X. Zhang, M. Guo, H. Wang, M. Wang, A question-answering assistant over personal knowledge graph, in: Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 2708–2712. URL: https://doi.org/10.1145/3626772.3657665. doi:10.1145/3626772.3657665.

[2] L. Shi, Z. Tang, N. Zhang, X. Zhang, Z. Yang, A Survey on Employing Large Language Models for Text-to-SQL Tasks, arXiv e-prints (2024) arXiv:2407.15186. doi:10.48550/arXiv.2407.15186. arXiv:2407.15186.

[3] B. Wang, R. Shin, X. Liu, O. Polozov, M. Richardson, RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers, in: D. Jurafsky, J. Chai, N. Schluter, J. Tetreault (Eds.), Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 7567–7578. URL: https://aclanthology.org/2020.acl-main.677/. doi:10.18653/v1/2020.acl-main.677.

[4] R. Cao, H. Zhang, H. Xu, J. Li, D. Ma, L. Chen, K. Yu, Astormer: An AST structure-aware transformer decoder for text-to-sql, CoRR abs/2310.18662 (2023). URL: https://doi.org/10.48550/arXiv.2310.18662. doi:10.48550/ARXIV.2310.18662. arXiv:2310.18662.

[5] Y. Zhou, Y. He, S. Tian, Y. Ni, Z. Yin, X. Liu, C. Ji, S. Liu, X. Qiu, G. Ye, H. Chai, r^3-NL2GQL: A model coordination and knowledge graph alignment approach for NL2GQL, in: Y. Al-Onaizan, M. Bansal, Y.-N. Chen (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2024, Association for Computational Linguistics, Miami, Florida, USA, 2024, pp. 13679–13692. URL: https://aclanthology.org/2024.findings-emnlp.800/. doi:10.18653/v1/2024.findings-emnlp.800.

[6] Y. Liang, K. Tan, T. Xie, W. Tao, S. Wang, Y. Lan, W. Qian, Aligning large language models to a domain-specific graph database for nl2gql, in: Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, CIKM '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 1367–1377. URL: https://doi.org/10.1145/3627673.3679713. doi:10.1145/3627673.3679713.

[7] Q. Zhang, H. Chen, J. Dong, W. Li, F. Huang, X. Huang, Structure-guided large language models for text-to-SQL generation, 2025. URL: https://openreview.net/forum?id=ReKWjKvkJE.

[8] L. Netz, J. Reimer, B. Rumpe, Using grammar masking to ensure syntactic validity in llm-based modeling tasks, in: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS Companion '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 115–122. URL: https://doi.org/10.1145/3652620.3687805. doi:10.1145/3652620.3687805.

[9] S. Geng, M. Josifoski, M. Peyrard, R. West, Grammar-constrained decoding for structured NLP tasks without finetuning, in: H. Bouamor, J. Pino, K. Bali (Eds.), Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023. URL: https://aclanthology.org/2023.emnlp-main.674.

[10] S. Ugare, T. Suresh, H. Kang, S. Misailovic, G. Singh, Syncode: Llm generation with grammar augmentation, 2024. arXiv:2403.01632.

[11] B. T. Willard, R. Louf, Efficient guided generation for large language models, arXiv preprint arXiv:2307.09702 (2023).

[12] Y. Dong, C. F. Ruan, Y. Cai, R. Lai, Z. Xu, Y. Zhao, T. Chen, Xgrammar: Flexible and efficient structured generation engine for large language models, 2025. URL: https://arxiv.org/abs/2411.15100. arXiv:2411.15100.

[13] L. Beurer-Kellner, M. Fischer, M. Vechev, Guiding llms the right way: fast, non-invasive constrained generation, in: Proceedings of the 41st International Conference on Machine Learning, ICML'24, JMLR.org, 2024.

[14] Y. Zhang, J. Chi, H. Nguyen, K. Upasani, D. M. Bikel, J. E. Weston, E. M. Smith, Backtracking improves generation safety, in: The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025, OpenReview.net, 2025. URL: https://openreview.net/forum?id=Bo62NeU6VF.