

Graph Queries from Natural Language using Constrained Language Models and Visual Editing

Benedikt Kantz

*Institute of Visual Computing
Graz University of Technology
Graz, Austria
benedikt.kantz@tugraz.at*

Stefan Lengauer

*Institute of Visual Computing
Graz University of Technology
Graz, Austria
s.lengauer@tugraz.at*

Kevin Innerebner

*Institute of Human-Centred Computing
Graz University of Technology
Graz, Austria
innerebner@tugraz.at*

Elisabeth Lex

*Institute of Human-Centred Computing
Graz University of Technology
Graz, Austria
elisabeth.lex@tugraz.at*

Peter Waldert

*Institute of Visual Computing
Graz University of Technology
Graz, Austria
peter.waldert@tugraz.at*

Tobias Schreck

*Institute of Visual Computing
Graz University of Technology
Graz, Austria
tobias.schreck@tugraz.at*

Abstract—Querying knowledge bases using ontologies is usually performed using dedicated query languages, question-answering systems, or visual query editors for Knowledge Graphs (KGs). We propose a novel approach that enables users to query the knowledge graph by specifying prototype graphs using Natural Language (NL) and visually editing them. This approach enables non-experts to formulate queries without prior knowledge of the ontology and specific query languages. Our approach converts NL queries to these prototype graphs by utilizing a two-step constrained Language Model (LM) generation based on semantically similar features within an ontology. The resulting prototype graph serves as the building block for further user refinements within a dedicated visual query builder. Our approach consistently generates a valid SPARQL query within the constraints imposed by the ontology, without requiring any additional corrections to the syntax or classes and links used. Unlike related LM approaches, which often require multiple iterations to fix invalid syntax, non-existent classes, and non-existent links, our approach achieves this consistently. We evaluate the performance of our system using graph retrieval on synthetic queries, comparing multiple metrics, models, and ontologies. We further validate our system through a preliminary user study. By utilizing our constrained pipeline, we show that the system can perform efficient and accurate retrieval using more efficient models compared to other approaches.

Index Terms—Ontologies, graph retrieval, natural language queries, visual query interfaces

I. INTRODUCTION

Ontologies are a foundational approach to represent the graph schema for KGs and enable knowledge transfers, exploration, and representation for further processing [1]. These ontologies and belonging KGs can achieve significant sizes and are, therefore, challenging for users unfamiliar with the knowledge domain to explore and search. One example of

This work is partially supported by the HEREDITARY Project, as part of the European Union’s Horizon Europe research and innovation programme under grant agreement No GA 101137074, the Austrian Science Fund (FWF) 10.55776/COE12, Cluster of Excellence Bilateral Artificial Intelligence and the FFG HybridAir project #FO999902654.

such an extensive collection of knowledge is DBpedia [2], which compiles linked articles into hundreds of classes and connects them through their various properties within the class hierarchy. Such a rich ontology can pose challenges for users building queries.

The ontologies are usually queried using specialized query languages, such as SPARQL [3], which can pose an additional barrier of entry for users seeking to retrieve structured knowledge from such systems. This paper, therefore, proposes a novel querying strategy, which maps relational queries in NL to prototype graph representations, which can then be further adjusted by the user to desired retrieval criteria. The system is tuned for NL queries formulated as relations, differing from traditional opaque question-answering approaches, which typically do not visualize the generated queries or their processing, nor allow edits to the query.

Our system extracts the prototype graph structure from the query using the capabilities of LMs to extract information from NL input – even in cases where there is no exact match for the required information within the ontology. We achieve a valid prototype graph by constraining the LM output using a dynamically created grammar to adhere to the classes and links present within the ontology. This novel addition of the grammar to the query generation by the LM enables our system, in turn, to always return prototype graphs that are valid within the context of the used ontology. Our system, furthermore, allows the user to refine and adjust the prototype graph in a visual editor, enabling the correction of mistakes the LM might make. Within this editor, users can refine their queries or extend them to more complex queries, allowing them to edit their queries without modifying the SPARQL query directly or any other intermediate representation in code.

We evaluate the prototype graph extraction performance of our approach using a synthetic benchmark, consisting of sampled sub-graphs and corresponding LM-generated NL queries. This synthetic benchmark is tested for alignment with

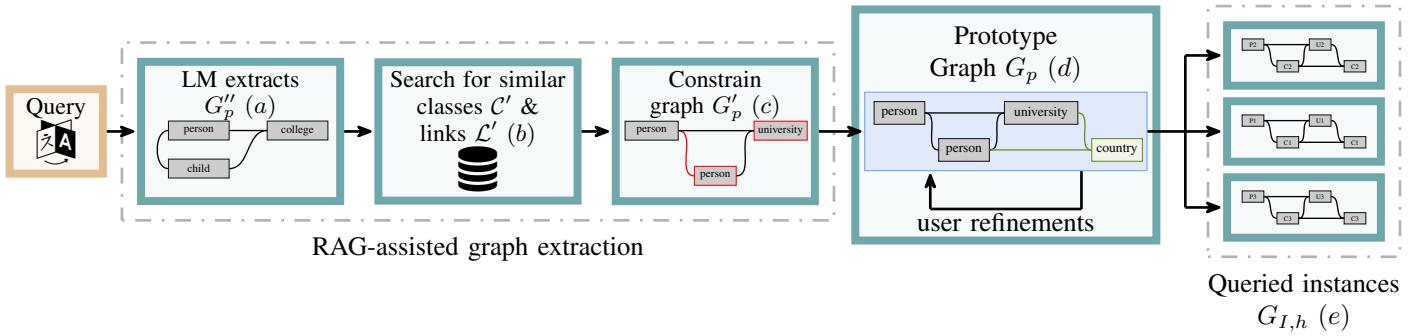


Fig. 1: Our query extraction process using LMs, using the query example “a person and the child of a person have the alma mater of the same university”. We transform the NL query into a prototype graph using a constrained LM. The graph is first approximated using a LM (a), where the generated classes and links might not match the ontology yet. This initial guess of the LM is used to search for semantically similar relations (b). With the subset of all possible links and nodes, the graph is extracted again and corrected for possible errors (c), resulting in a graph that adheres to the ontology. The resulting graph can be edited (d), e.g., an additional constraint for a country can be added. The resulting prototype graph can be used to perform queries over a KG to retrieve instances (e).

human query examples using our synthetic query generation framework. We extend this evaluation to test the integrated system within a user study. This study enables us to compare this combined system of NL querying and visual editing with existing visual query editors, as well as formal question-answering benchmarks. Our retrieval approach, furthermore, does not require any metadata about the ontologies or query examples, unlike other systems [4].

II. RELATED WORK

Previous efforts to map queries presented in NL towards complex, constrained query languages [4]–[6] focused on retrieving and generating SPARQL queries directly, resulting in either highly complex systems, or a retrieval process with many iterative refinement steps.

Lei et al. [5] map a NL query onto a particular, tailored, *Ontology Query Language*, used as an intermediate representation, and then converted to SQL. They use the ontology to find relevant terms in the NL query and map them to elements of the ontology. This approach limits the output to specific instances, hindering both extensibility and exploration. *SPARKLIS* [6] approaches the mapping from NL to a SPARQL query through a mixture of highly constrained NL and visual exploration. The constraints placed upon the queries help to adhere to the graph schema but could hinder more straightforward exploration.

More typically, KGs are queried using SPARQL [3], which can be generated with LMs [4], [7], [8]. Meyer et al. [7] have shown that the out-of-the-box performance of multiple proprietary LMs is lacking for direct generation of SPARQL queries from NL queries. They demonstrate that adding the ontology, in textual representation, to the query enhances the generation. The generation is further improved if only relevant classes and relations are provided. Similarly, Emonet et al. [4] improve upon these results by targeting large-scale federated KGs. They develop a Retrieval Augmented

Generation (RAG) system that automatically augments the LM input with relevant ontology classes and manually created example queries. Finally, Liu et al. [8] introduce SPINACH, a question-answering LM agent for Wikidata, that iteratively performs actions: searching Wikidata for entities, properties, or example queries, and executing SPARQL queries on demand. Effectively, these methods highlight that incorporating the ontology is essential to improve generated SPARQL queries. Nevertheless, these methods require the LMs to generate a syntactically and semantically correct SPARQL query, but—due to syntax errors or hallucination of properties—can require feedback error messages to fix the query iteratively.

On the other hand, the visual querying approaches focus on providing approachable systems to facilitate the use of knowledge graphs for non-experts. *GRAPHITE* [9] and *VIS-AGE* [10] both employ visual editing of the prototype graphs to foster the exploration of KGs with a focus on fuzzy matching, query suggestions, and fast retrieval times. *RDF Explorer* [11] approaches this task similarly by providing a graph editor, enabling the creation of graph examples to retrieve matching instances from the knowledge graph. The system provides users with query expansion options throughout the application, displaying dynamic results as queries are built. The authors relate their work to previous query builders, notably *Smeagol* [12]. This alternative follows similar exploration and retrieval paradigms but does not offer a comparable number of SPARQL features. *KGVQL* [13] defines a novel visual query language to ease the transformation between the visual querying and result set, at the cost of disregarding explorative approaches, favoring the proposed transformation approach to convert between data examples and queries. *Rhizomer* [14] approaches the exploration of knowledge bases by providing different user interfaces to explore only at the top-level, graph-level, or only at the instance level of a single type. *Sparnatural* [15] simplifies many of these exploratory approaches into

a tree-based approach.

In comparison to these works, our method utilizes the ontology to constrain the LM output, thereby creating a prototype graph that can be directly mapped to a valid SPARQL query. This intermediate graph eliminates syntax errors and hallucinations while enabling the usage of smaller models ($\leq 8B$ parameters) and removing the need for feedback error messages. The system is also robust against semantic mismatches between the query and schema, allowing users to formulate their queries more freely. We also seamlessly integrate this NL querying system with a visual interface, enabling users to adjust and refine their query after the mapping phase has been completed.

We, furthermore, did not find any applicable benchmark dataset within the related works analyzed within this paper. The usual system of using a Q&A system or direct query generation framework is not applicable to our goal of mapping from a NL query to a graph representation, which can be evaluated in a more direct way using well-established scores like the F_1 score or Graph Edit Distance (GED) [16]. The evaluation of our system, therefore, builds upon a synthetic evaluation pipeline, combined with alignment tests and a small-scale user study to confirm our results.

III. METHODOLOGY

Our KG retrieval approach builds on the notion of graph extraction and graph instance retrieval. To realize this notion, we require the graph extraction from NL as outlined in Figure 1. The input to our pipeline is a NL query, returning a prototype graph G_p with nodes N_p and edges E_p . The graph extraction performance is evaluated using a synthetic dataset, generated with our query generation pipeline. The synthetic dataset is shown to be representative of real results through a comparison of using both synthetic and human-written queries on a subset of queries. We provide further details to foster reproducibility in Appendix Section C.

A. Graph Extraction

Our knowledge graph retrieval system builds upon the notion of a prototype graph $G_p := (N_p, E_p)$ that serves as the blueprint for all retrieved instances. This graph representation is similar to Basic Graph Pattern (BGP), with the extension of adhering to the schema imposed by the ontology. The subgraph G_p of the ontology consists of the nodes N_p , each one an instance of the classes \mathcal{C} , and edges E_p , each one a link of link types \mathcal{L} . The multiset of nodes can contain a class multiple times, and a link can only span the allowed end and start types (within the class hierarchy), i.e.

$$\begin{aligned} N_p &:= \{n_1, \dots, n_m\} \subseteq \mathcal{C}, \\ E_p &\subseteq \{e_i = (n_t, l_j, n_h) \mid \text{subtypeof}(n_t, \text{fromtype}(l_j)), \\ &\quad \text{subtypeof}(n_h, \text{totype}(l_j))\} \subseteq (N_p \times \mathcal{L} \times N_p). \end{aligned}$$

We then retrieve the instance graphs $G_{I,h} := (O_{I,h}, P_{I,h})$ from the knowledge base with objects from all vertices or

objects \mathcal{O} and predicates from all predicates \mathcal{P} in the KG that match the prototype graph G_p , i.e.

$$\begin{aligned} O_{I,h} &:= \{o_i \in \mathcal{O} \mid \text{typeof}(o_i) \in N_p\}, \\ P_{I,h} &:= \{p_i = (o_t, l_j, o_h) \mid o_t, o_h \in \mathcal{O}_{I,h}, \text{typeof}(o_t) \in N_p, \\ &\quad \text{typeof}(o_h) \in N_p, p_i \in \mathcal{P}, l_j \in \mathcal{L}\}. \end{aligned}$$

Therefore, this retrieval approach requires a prototype graph G_p that matches the types within the possible classes \mathcal{C} and links \mathcal{L} to provide meaningful results. Our graph extraction pipeline from the NL achieves this constrained retrieval using a multistep approach illustrated in Figure 1. This approach first extracts an unconstrained graph G''_p from the NL prompt using the structured output of a LM [17], which serves as the basis for retrieval of semantically similar and existing classes \mathcal{C} and links \mathcal{L} . These are then used to perform another round of structured generation to get G'_p , which is refined to the final prototype graph G_p that can be used to retrieve instance graphs $G_{I,h}$.

1) *Graph from NL*: This first unconstrained graph generation step is required as the basis for further querying of possible classes \mathcal{C} and links \mathcal{L} . The LM output is nevertheless restrained to return only a specific JSON schema through GGML Backus-Naur Form (GBNF) [17], [18]. This measure enforces a consistent output that can be parsed and used for further processing. These constraints allow us to construct the intermediate graph G''_p , which has no constraints regarding the ontology, i.e., \mathcal{L} and \mathcal{C} are open. While we could constrain the graph types to the whole ontology at this step, we have no way to enforce the structural correctness of the graph over the outgoing link types, increasing the probability of an invalid graph.

2) *Constraints from Graph*: The unconstrained graph is used to retrieve candidates for the next generation step. This retrieval is performed for each node $n_i \in N_p$ and edge $e_j \in E_p$ using sentence embeddings of the node and link description. These embeddings are used to retrieve the top k most similar results in terms of cosine distance from all classes \mathcal{C} and links \mathcal{L} . The LM generation is then further constrained to only include these results. This retrieval of semantically similar links results in the subset of classes $\mathcal{C}' \subseteq \mathcal{C}$ and links $\mathcal{L}' \subseteq \mathcal{L}$.

3) *Constrained Graph from NL*: Finally, the prototype graph G_p is generated by providing the LM with the same instruction as in the first step, but with additional constraints placed upon the output generation through GBNF [18]. These use the additional information of the possible candidate classes \mathcal{C}' and links \mathcal{L}' from the previous step. This limited set of only semantically similar classes enables the LM still to express any graph from the NL query while adhering to the relevant query constraints. The LM-generated graph G'_p may contain invalid or flipped links as we cannot enforce a valid graph structure on the output. This problem is mitigated by the previous step of only using the ontology subsets and by cleaning the graph using two rules. The first one exchanges the direction of the edge, essentially swapping the nodes $(n_t, l_j, n_h) \mapsto (n_h, l_j, n_t)$, if the types are flipped, i.e., the

link l_j may not go from n_t to n_h , but from n_h to n_t . Our second rule discards any invalid links from the graph if they violate the type constraints.

B. Synthetic Evaluation Methodology

The described graph retrieval system is evaluated using synthetically generated queries from a sampled prototype graph $G_{p,s}$. This graph is used to generate queries in NL using either a LM that is prompted with the graph as input structure or a template-based query generation. We additionally validate our query generation methodology against queries generated by humans by comparing the resulting evaluation metrics. The NL prompt is, in turn, used to extract the prototype graph G_p using the method from above. Finally, the sampled and extracted graphs are compared and evaluated for similarity. This evaluation system does not incorporate user refinements for adjusting the output graph, as the methodology evaluates only the directly returned prototype graph G_p . We, furthermore, compare the query extraction of our system to the output of the LM without any alignments, i.e. the first unconstrained output of our system, in an ablation study performed for all evaluated settings.

1) *Graph Sampling*: The first step in our synthetic evaluation pipeline is the sampling of prototype graphs $G_{p,s}$ from the ontology. The sampling is based on probabilities derived from the instance counts of the links. We additionally select classes that are lower in the class hierarchy using a similar probabilistic approach. Finally, further links are added based on a random choice of node and a similar probabilistic selection. This probabilistic sampling is only performed for ontologies with sufficient instances, as indicated in Section III-B4. The classes and links are sampled uniformly for the other cases. Additional graph sampling details can be found in Section A.

2) *Generation of the Query*: Next, the NL query is generated synthetically from the previously sampled graph $G_{p,s}$ using a LM, more specifically the Hermes 3 Llama 3.2 3B model [19]. The LM is presented with a textual representation of the types of classes and links within the graph and prompted, using a one-shot approach, to generate the queries. We also employ a template-based query generation approach to prevent potential information leaks and evaluate simpler queries [20]. Additionally, we create 12 human-written queries from the sampled graphs to validate our evaluation methodologies for queries used in practice.

3) *Scoring the Graphs*: Finally, the prototype graph G_p can be extracted from the synthetic query using our graph extraction methodology and compared to the ground truth sampled graph $G_{p,s}$. This evaluation employed two scoring methodologies: one for the retrieval performance of the nodes N_p and relations E_p , and another for the graph layout. First, the similarity between the retrieved nodes N_p and sampled nodes $N_{p,s}$ is compared using the F_1 -score over the sets. This set-based $F_{1,rel}$ -score uses the true positives $TP = |N_p \cap N_{p,s}|$, false positives $FP = |N_p - N_{p,s}|$ and false negatives $FN = |N_{p,s} - N_p|$ rates from set intersections and

differences. The $F_{1,rel}$ of the relations is computed using the same scheme using E_p and $E_{p,s}$.

Second, the graph similarity is computed using the GED [16], which employs a stricter notion of similarity, requiring not only isomorphism between the graphs but also the same node and link types for the graphs. To achieve a comparable score to the F_1 scores and between different graph sizes, we weigh the distance by the number of nodes and links and invert it, giving the normalized GED score

$$GED_s = 1 - \frac{GED}{\max\{|N_p|, |N_{p,s}|\} + \max\{|E_p|, |E_{p,s}|\}}.$$

Both evaluation methodologies provide scores for a single query example. We therefore reduce them to single values by averaging the scores over the different queries and evaluation settings.

4) *Evaluated Models and Ontologies*: The evaluation of this work relies heavily on the use of LMs for graph retrieval, semantic retrieval, and constrained retrieval. We use the Hermes models [19] (3B, 8B, and 70B parameter sizes) for generative retrieval tasks due to their fine-tuned capabilities for structured output, comparing them to two Qwen2.5 (32B parameters) models (Instruct and Code fine-tuned) [21]. The Stella 400M [22], [23] model is used for all semantic retrieval tasks. Further implementation details can be found in Section C.

We evaluate our approach on four ontologies, specifically

- DBpedia [2], an ontology and KG containing mapped information from Wikidata;
- Yago 4 [24], a similarly curated version of Wikidata on the schema.org classes;
- Brainteaser Ontology (BTO) [25], a smaller ontology and KG focusing on brain-related diseases; and
- UniProt [26], a similarly focused ontology and KG containing vast amounts of protein sequences and their functional information.

Only DBpedia [2] and Yago 4 [24] use the probabilistic sampling based on instance count, as the other two ontologies do not provide such a large KG to enable good sampling, using the uniform sampling approach instead.

C. Constraining the LM

The LMs are constrained to a schema specified to a grammar whenever we retrieve any graph using language generation. To this end, we use `llama-cpp-agent`¹, which can constrain the LM to only output a specific model using a predefined grammar. The first prototype graph G_p'' uses a static schema and grammar. In the next generation step, the grammar is adapted to the specific, similar relations found utilizing the output of the first round. The updated grammar is then used to constrain the output only to contain valid types and links, which can be used to generate the correct prototype graph G_p . The graph G_p can be converted into a SPARQL query if the user requires it for further use, as shown in Section B.

¹<https://llama-cpp-agent.readthedocs.io/>

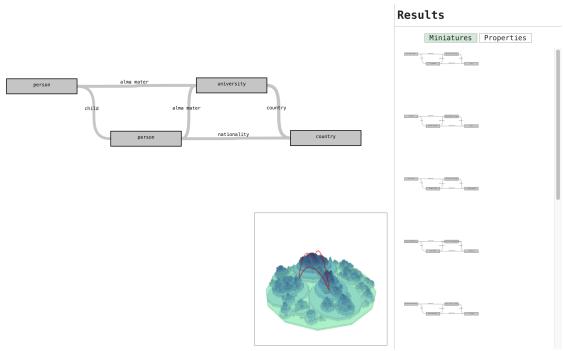


Fig. 2: OnSET user interface [27] showing the query for the example from Figure 1. The user can edit the prototype graph G_p on the left, view the connected nodes at the bottom using a circle-packing visualization of the ontology, and inspect the instances $G_{I,h}$ directly on the right within the interface.

D. User Interface

Additionally, we allow the users to refine their initial queries using a node-based editor, where links and nodes can be added to or removed from the graph, all within the constrained link set \mathcal{L} , and class set \mathcal{C} . This refinement can be helpful if our pipeline fails to find the correct graph or if the users want to refine their search without an additional text prompt. We support a similar refinement process as the second step in our pipeline, which involves performing semantic search for link retrieval. We use the cosine similarity of sentence embeddings [23] to compare possible new links that the user can search through when adding these. The query editor is based on the prior work within the Ontology and Semantic Exploration Toolkit (OnSET) [27] and shown in Figure 2.

Furthermore, we transparently display how the queries are built using the LM through a similar flow to that shown in Figure 1. This visualization should hold our system accountable for any mistakes and errors that occur during processing and may guide the user towards specifying more precise queries or exploring other querying avenues.

E. User Study

We additionally conduct a preliminary user study to support our claims on usability and improvements in query formulation. Our study focuses on two aspects of the user’s experience:

- 1) The user’s ability to formulate the correct queries given increasingly complex tasks, with the option to adjust queries using our proposed user interface.
- 2) The effectiveness of the integrated NL query interface and visual editor.

To validate our claims, we first task the users with completing three increasingly complex queries, as shown in Table I. Each user watches a 2-minute introductory video and has 20 minutes to perform all four tasks. We time the completion of each task and note whether it has the correct number of nodes and links, as well as the correct node types, link types, and constraints. After they complete all tasks or run out of

TABLE I: User tasks on the DBpedia KG.

No.	Task	k
0	Find all works where the author is a hockey player.	2
1	Persons that are authors of a work and are gold medalists in a sports event.	3
2	Ships that have their home port in a place in the country United States and persons who have the same place as their death place.	4
3	A work that is the opening theme for a TV show is composed by a person and that person has a child. The person has the alma mater of an University.	5

time, the user is presented with a System Usability Score (SUS) [28] evaluation questionnaire to estimate their load on our system. We compare our system to the RDF Explorer, which is evaluated using a similar strategy [11].

IV. RESULTS

We demonstrate our system’s capabilities to retrieve the correct graph from the query using our synthetic graph generation and evaluation pipeline. We evaluate the system at three different node sample counts, $k \in \{2, 3, 5, 7\}$, to model varying degrees of complexity in the queries. Each node sample count k is sampled for 128 synthetic queries. Additionally, we use five different open-weight models to test the dependence on model size and type. Our evaluation in Figure 3 shows that we can faithfully recover the prototype graph from the NL query. While we cannot achieve a perfect recovery of all nodes and relations in all settings, we accomplish a F_1 score on the node retrieval on DBpedia across all models of approximately 0.7 throughout the different levels of complexity. The correct relations are retrieved at an even higher F_1 score of roughly 0.8 for the various ontologies and settings. Similarly, the GED score GED_s shows that the graph structure can be recovered quite well for most of the smaller graph sizes and drops slightly for the larger, more complex queries, demonstrating that our approach is most useful for smaller graphs. Our evaluation, furthermore, shows that using a larger model is only slightly beneficial for our use case. The system performs similarly across all model sizes and types, suggesting that even smaller models can reconstruct the prototype graph quite well due to the constraints we place on the output.

The evaluation also shows that the constraint and alignment step to the ontology, including our corrections, is essential to retrieve the correct graph. This is evident by the ablation study performed by comparing the *raw* output of the LM to the *aligned* outputs of our fill pipelines. The system achieves indeed higher scores in the *aligned* results, and can be observed for almost all combinations of query complexities, LM models, and ontologies.

Finally, we compare our different query generation methods in Table II by evaluating two query complexities $k = \{3, 5\}$ and comparing the scores. The LM-based generation method performs similarly to the human-generated queries. This comparative evaluation, additionally, shows that the template queries perform better than the compared LM-generated ones.

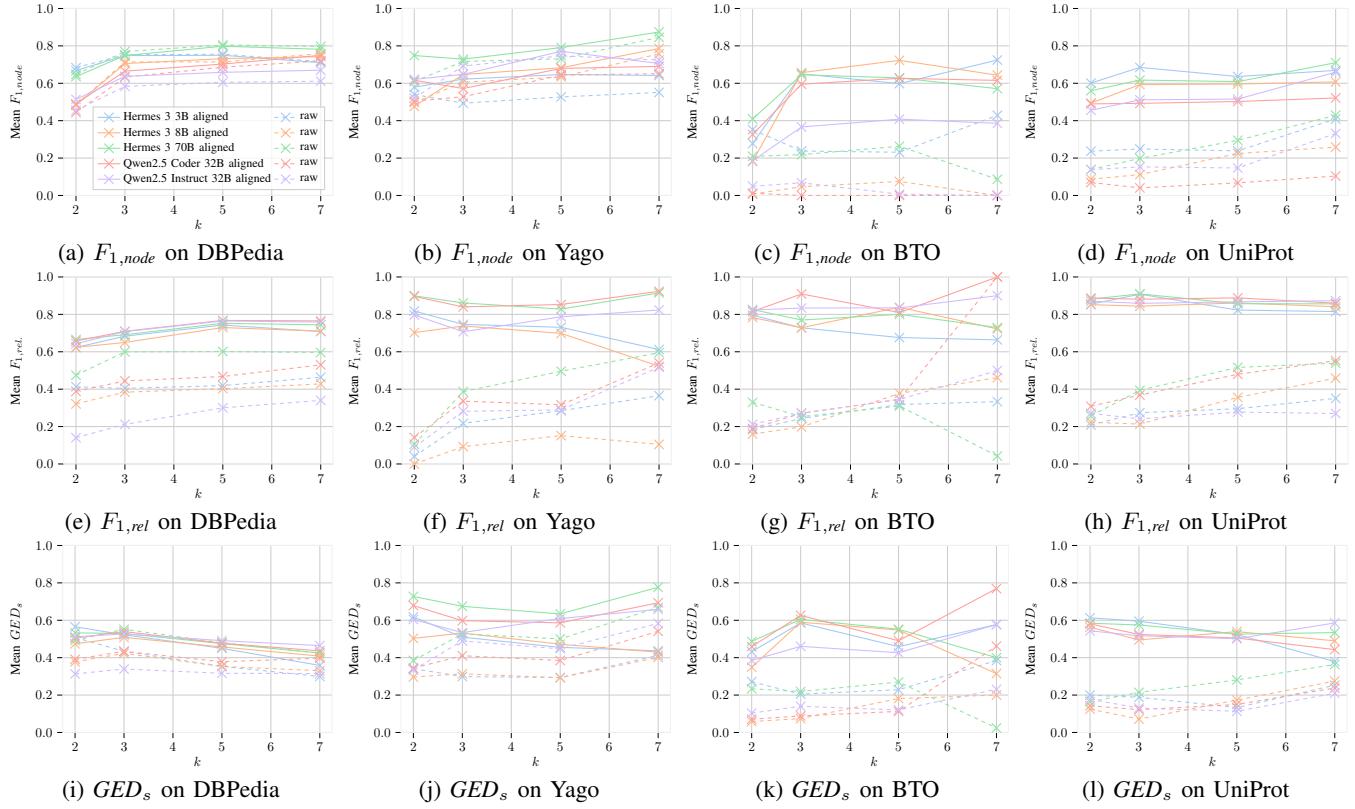


Fig. 3: $F_{1,\text{node}}$, $F_{1,\text{rel}}$, and GED_s on four different ontologies, comparing different models and node amounts k , and the *raw* and *aligned* (constrained) output.

TABLE II: Comparison of query generation methods for Hermes 3B on DBpedia.

k	Query Origin	Stage	Mean $F_{1,\text{node}}$	Mean GED_s
3	human	aligned	0.63	0.40
		raw	0.52	0.24
	llama	aligned	0.71	0.46
		raw	0.68	0.36
	templated	aligned	0.81	0.57
		raw	0.79	0.54
	5	human	0.69	0.22
		raw	0.90	0.27
		llama	0.66	0.34
		raw	0.67	0.25
	templated	aligned	0.83	0.48
	templated	raw	0.74	0.45

These results indicate that the LMs perform better with simpler formatted queries, such as the template-generated queries.

A. User Study

We also inspect the preliminary results of our user study shown in Table III, which involved $n = 11$ participants. We observe that our average correct completion rate (“Success Rate”) is high throughout the increasingly complex tasks, compared to the study on *RDFExplorer* [11]. They report that their correctness decreases, possibly due to time constraints. Within the survey of our tool, however, no user took longer than the

TABLE III: Task results for the OnSET user study on DBpedia with $n = 11$.

Task	Success Rate	Time (mm:ss)
Task 0	0.73	03:46
Task 1	0.91	01:12
Task 2	0.73	04:49
Task 3	1.00	02:05

20-minute time frame, and we observed lower success rates only for the first and third tasks. These tasks might have been more challenging because the LM did not directly provide the correct prototype graph. The most complex task, however, had the highest correctness rate compared to the others in our study. These results, in contrast to the observations of the *RDFExplorer*, indicate that our system performs consistently across varying levels of difficulty and primarily relies on the LM output. The evaluation of our SUS questionnaire resulted in a score of 71.4.

V. CONCLUSION

We introduce a novel querying system in this paper to aid users in exploring ontologies and generating queries from NL. The interface uses only NL as the input, providing the users with a fuzzy interface to explore an otherwise strict system of classes and links. We achieve this translation from fuzzy

input to the strict notion using a query processing pipeline that heavily utilizes LMs for generating the prototype graph, performing semantic similarity search, and constrained graph generation. We provide a prototype graph as an intermediate output from this pipeline. This prototype graph can be refined through a node-based editor to better reflect the users' information needs if the LM failed to map the NL query completely, or the user has a more specific information need.

We evaluate our system on a set of synthetic queries with varying levels of complexity. These should reflect users' varying information needs and queries while evaluating our system for more complex tasks. This evaluation demonstrates that the extracted prototype graphs accurately represent the sampled graphs, particularly for smaller graphs. This performance level, especially at the smaller graph sizes, should provide a robust system that adheres to the users' initial interests. We demonstrate that this level of performance can be achieved with smaller models.

In contrast, existing systems struggle with these smaller models and require significantly more powerful, and thus more expensive, models. They, furthermore, often require iterative systems to retrieve syntactically correct queries, whereas our system can provide valid queries using our two-step retrieval process. We furthermore validate our approach by comparing the synthetic queries generated by a LM and template system from the sampled graphs to a small set of manually written queries. These evaluations show that the LM-generated queries do indeed adhere to the manually written queries, thereby strengthening our query evaluation approach.

We additionally conducted a small-scale user study to assess our systems' performance with respect to usability and task completion time, where we observed high rates of correctness and fast completion times compared to existing systems. This suggests that combining a precise and efficient LM system with a visual query editor can reduce the time spent on creating queries while improving the correctness of the results compared to prior works.

VI. FUTURE WORK

The introduced system builds a prototype graph based on NL input, enabling users to explore and query knowledge graphs effectively. Future versions could, however, enhance the robustness of retrieving the prototype graph from NL queries for larger and more complex queries by utilizing more sophisticated models or fine-tuning smaller, more specialized models for the graph retrieval task at hand, thereby achieving even more precise initial graph responses.

Another avenue for further improvement in this system is the addition of constraints to the graph generated by the LMs. While we can constrain the properties (e.g., the age, name, or height of a person) within the user interface, we do not provide a way for the LM to add this to the generated query. These, however, are an integral part of the KG and the information they contain. We intend to extend the LM output and graph to include these filters on the edges and formalize these structures to get a constrained output, including these constraints.

The user study will also be extended to include more users, with additional questionnaires and possibly more challenging tasks, to facilitate a more comprehensive comparison with other systems using visual query builders.

REFERENCES

- [1] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?" *Int. J. Hum. Comput. Stud.*, vol. 43, no. 5, pp. 907–928, Nov. 1995.
- [2] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web*, vol. 6, pp. 167–195, 2015. [Online]. Available: <https://api.semanticscholar.org/CorpusID:1181640>
- [3] A. Seaborne, R. Taelman, G. Williams, O. Hartig, and T. P. Tanon, "SPARQL 1.2 query language," W3C, W3C Working Draft, Dec. 2024, <https://www.w3.org/TR/2024/WD-sparql12-query-20241227>.
- [4] 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," 2024. [Online]. Available: <https://arxiv.org/abs/2410.06062>
- [5] C. Lei, F. Özcan, A. Quamar, A. R. Mittal, J. Sen, D. Saha, and K. Sankaranarayanan, "Ontology-based natural language query interfaces for data exploration." *IEEE Data Eng. Bull.*, vol. 41, no. 3, pp. 52–63, 2018.
- [6] S. Ferré, "SPARKLIS: An Expressive Query Builder for SPARQL Endpoints with Guidance in Natural Language," *Open Journal Of Semantic Web*, vol. 0, 2017. [Online]. Available: <https://inria.hal.science/hal-01485093>
- [7] L.-P. Meyer, J. Frey, F. Brei, and N. Arndt, "Assessing sparql capabilities of large language models," 2024. [Online]. Available: <https://arxiv.org/abs/2409.05925>
- [8] S. Liu, S. Semnani, H. Triedman, J. Xu, I. D. Zhao, and M. Lam, "SPINACH: SPARQL-Based Information Navigation for Challenging Real-World Questions," *ACL Anthology*, pp. 15 977–16 001, Nov. 2024.
- [9] D. H. Chau, C. Faloutsos, H. Tong, J. I. Hong, B. Gallagher, and T. Eliassi-Rad, "Graphite: A visual query system for large graphs," in *2008 IEEE International Conference on Data Mining Workshops*, 2008, pp. 963–966.
- [10] R. Pienta, A. Tamersoy, A. Endert, S. Navathe, H. Tong, and D. H. Chau, "Visage: Interactive visual graph querying," in *Proceedings of the International Working Conference on Advanced Visual Interfaces*, ser. AVI '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 272–279.
- [11] H. Vargas, C. Buil-Aranda, A. Hogan, and C. López, "Rdf explorer: A visual sparql query builder," in *The Semantic Web – ISWC 2019*, C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, and F. Gandon, Eds. Cham: Springer International Publishing, 2019, pp. 647–663.
- [12] A. Clemmer and S. Davies, "Smeagol: A "specific-to-general" semantic web query interface paradigm for novices," in *Database and Expert Systems Applications*, A. Hameurlain, S. W. Liddle, K.-D. Schewe, and X. Zhou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 288–302.
- [13] P. Liu, X. Wang, Q. Fu, Y. Yang, Y.-F. Li, and Q. Zhang, "KGSQL: A knowledge graph visual query language with bidirectional transformations," *Knowledge-Based Systems*, vol. 250, p. 108870, Aug. 2022.
- [14] R. García, J.-M. López-Gil, and R. Gil, "Rhizomer: Interactive semantic knowledge graphs exploration," *SoftwareX*, vol. 20, p. 101235, Dec. 2022.
- [15] T. Francart, "Sparnatural: a visual knowledge graph exploration tool," in *European Semantic Web Conference*. Springer, 2023, pp. 11–15.
- [16] Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, and P. Martineau, "An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems," in *4th International Conference on Pattern Recognition Applications and Methods 2015*, Lisbon, Portugal, Jan. 2015.
- [17] M. X. Liu, F. Liu, A. J. Fiannaca, T. Koo, L. Dixon, M. Terry, and C. J. Cai, "“we need structured output”: Towards user-centered constraints on large language model output," in *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, ser. CHI '24. ACM, May 2024, p. 1–9. [Online]. Available: <http://dx.doi.org/10.1145/3613905.3650756>

- [18] llama.cpp authors, “Gbnf guide,” Feb 2025, [Online; accessed 2025-02-11]. [Online]. Available: <https://github.com/ggermanov/llama.cpp/blob/b9ab0a4d0b2ed19effec130921d05fb5c30b68c5/grammars/README.md>
- [19] R. Teknium, J. Quesnelle, and C. Guang, “Hermes 3 technical report,” 2024. [Online]. Available: <https://arxiv.org/abs/2408.11857>
- [20] S. Sannigrahi, T. Fraga-Silva, Y. Oualil, and C. Van Gysel, “Synthetic query generation using large language models for virtual assistants,” in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR 2024. ACM, Jul. 2024, p. 2837–2841.
- [21] Qwen, ;, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu, “Qwen2.5 technical report,” 2025. [Online]. Available: <https://arxiv.org/abs/2412.15115>
- [22] D. Zhang, J. Li, Z. Zeng, and F. Wang, “Jasper and stella: distillation of sota embedding models,” 2025. [Online]. Available: arxiv.org/abs/2412.19048
- [23] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: <https://arxiv.org/abs/1908.10084>
- [24] T. Pellissier Tanon, G. Weikum, and F. M. Suchanek, “YAGO 4: A reason-able knowledge base,” in *The Semantic Web - 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31-June 4, 2020, Proceedings*, ser. Lecture Notes in Computer Science, vol. 12123. Springer, 2020, pp. 583–596.
- [25] G. Faggioli, S. Marchesin, L. Menotti, G. M. D. Nunzio, G. Silvello, and N. Ferro, “The brainteaser ontology for als and ms clinical data,” 2024. [Online]. Available: <https://zenodo.org/doi/10.5281/zenodo.12789731>
- [26] T. U. Consortium, “Uniprot: the universal protein knowledgebase in 2025,” *Nucleic Acids Research*, vol. 53, no. D1, pp. D609–D617, 11 2024. [Online]. Available: <https://doi.org/10.1093/nar/gkae1010>
- [27] B. Kantz, K. Innerebner, P. Waldert, S. Lengauer, E. Lex, and T. Schreck, “Onset: Ontology and semantic exploration toolkit,” in *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR ’25. New York, NY, USA: Association for Computing Machinery, 2025, p. 3980–3984.
- [28] J. Brooke *et al.*, “Sus-a quick and dirty usability scale,” *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

APPENDIX

A. Graph sampling

We sample the ontology for queries using the procedure in Algorithm 1. The outlined algorithm should provide a broad variety of graph structures, classes, and relations to create a diverse set of queries for our evaluation, while being faithful to the interest of the users based on the probabilistic sampling.

B. Query Generation

We generate the SPARQL queries from our prototype graphs G_p directly by iterating over all links and nodes. Each link results in the link type and left and right nodes; each node adds the node as a class. Listing 1 shows a resulting query for an example query – the same query as in Figures 1 and 2.

C. Reproducibility

To ensure the reproducibility of our evaluation and enable further experiments on our user interface, we provide our code and parameters on github.com/Dakantz/OnSET. We performed the experiments on the smaller models ($\leq 32B$

Algorithm 1: Prototype graph $G_{p,s}$ sampling from the ontology using instance counts within the KG.

```

Require: classes  $\mathcal{C}$ , links  $\mathcal{L}$ , objects  $\mathcal{O}$ , predicates  $\mathcal{P}$ , samples  $k > 0$ , depth  $d > 0$ , maximum of nodes  $m_{\text{nodes}} > 0$ 
function DOWNGRADE NODE( $n_s$ )
   $C_{\text{sub}} \leftarrow$  subtypes of  $n_s$  up to depth  $d$ 
   $Pr_{\text{node}}(n_s) = \frac{|\{n_{I,h,i} \in \mathcal{C} \mid n_s = \text{typeof}(n_{I,h,i})\}|}{\sum_{n'_s \in C_{\text{sub}}} |\{n_{I,h,i} \in \mathcal{C} \mid n'_s = \text{typeof}(n_{I,h,i})\}|}$ 
  return  $n \sim Pr_{\text{node}}(n_s)$ 
end function
function DOWNGRADE LINK( $l = (n_{p,i}, n_{p,j}, l_{ij})$ )
   $n_{p,i,\text{sub}} \leftarrow$  DOWNGRADE NODE( $n_{p,i}$ )
   $n_{p,j,\text{sub}} \leftarrow$  DOWNGRADE NODE( $n_{p,j}$ )
  return  $l = (n_{p,i,\text{sub}}, n_{p,j,\text{sub}}, l_{ij})$ 
end function
 $\mathcal{L}_{\text{cand.}} \leftarrow$  top  $k$  links  $\mathcal{L}$  ▷ By instance count
 $P_{\text{link}}(l \in \mathcal{L}_{\text{cand.}}) = \frac{|\{e_{I,h,ij} \in \mathcal{P} \mid l = \text{typeof}(e_{I,h,ij})\}|}{\sum_{l' \in \mathcal{L}_{\text{cand.}}} |\{e_{I,h,ij} \in \mathcal{P} \mid l' = \text{typeof}(e_{I,h,ij})\}|}$ 
 $l_{\text{sel}} \sim P(l \in \mathcal{L}_{\text{cand.}})$ 
 $l_{\text{sub}} \leftarrow$  DOWNGRADE LINK( $l$ )
 $E_{p,s} \leftarrow \{l_{\text{sub}}\}$ 
 $N_{p,s} \leftarrow \{n_{p,0,\text{sub}}, n_{p,1,\text{sub}}\}$ 
while  $m_{\text{nodes}} > |N_{p,s}|$  do
   $side \sim \mathcal{U}\{left, right\}$ 
   $n_{\text{sel}} \sim \mathcal{U}(N_p)$ 
   $\mathcal{L}_{\text{cand.,next}} \leftarrow$  top  $k$  links attaching to  $side$ 
   $l_{\text{new}} \sim P_{\text{link}}(l \in \mathcal{L}_{\text{cand.,next}})$ 
   $l_{\text{new,sel}} \leftarrow$  DOWNGRADE LINK( $l_{\text{new}}$ )
   $E_{p,s} \leftarrow E_{p,s} \cup \{l_{\text{new,sel}}\}$ 
   $N_{p,s} \leftarrow N_{p,s} \cup \{\text{added node of } l_{\text{new,sel}}\}$ 
end while
output  $G_{p,s} = (N_{p,s}, E_{p,s})$ 

```

parameters) on a system with an RTX 4090, and the large-scale LM experiments on our university cluster on multiple Quadro RTX 8000s.

Listing 1: Example SPARQL query generated from the input “a person and the child of a person have the alma mater of the same university”.

```

SELECT DISTINCT ?person_1 ?person_2 ?
  university_1 WHERE {
    ?person_1 <http://dbpedia.org/ontology/child> ?person_2.
    ?person_1 <http://dbpedia.org/ontology/almaMater> ?university_1.
    ?person_2 <http://dbpedia.org/ontology/almaMater> ?university_1.
    ?person_1 a <http://dbpedia.org/ontology/Person>.
    ?person_2 a <http://dbpedia.org/ontology/Person>.
    ?university_1 a <http://dbpedia.org/ontology/University>.
  }

```