SPOT: Zero-Shot Semantic Parsing Over Property Graphs

Francesco Cazzaro[†], **Justin Kleindienst**[§], **Sofia Márquez Gomez**[§], **Ariadna Quattoni**[†§]
[†]Universitat Politècnica de Catalunya, Barcelona, Spain
[§]dMetrics, Jackson Wyoming USA
francesco.cazzaro@upc.edu, aquattoni@cs.upc.edu
{justin.kleindienst, sofia.marquez, ariadna.quattoni}@dmetrics.com

Abstract

Knowledge Graphs (KGs) have gained popularity as a means of storing structured data, with property graphs, in particular, gaining traction in recent years. Consequently, the task of semantic parsing remains crucial in enabling access to the information in these graphs via natural language queries. However, annotated data is scarce, requires significant effort to create, and is not easily transferable between different graphs. To address these challenges we introduce SPOT, a method to generate training data for semantic parsing over Property Graphs without human annotations. We generate tree patterns, match them to the KG to obtain a query program, and use a finite-state transducer to produce a proto-natural language realization of the query. Finally, we paraphrase the proto-NL with an LLM to generate samples for training a semantic parser. We demonstrate the effectiveness of SPOT on two property graph benchmarks utilizing the Cypher query language. In addition, we show that our approach can also be applied effectively to RDF graphs.

1 Introduction

The use of knowledge graphs (KGs) is expanding as a means to store general world knowledge as well as domain-specific knowledge (Abu-Salih, 2021; Li et al., 2024a). In particular, property graphs (PGs) have seen growing adoption in industrial settings. This is mainly because they enable the specification of semantically rich data models for organizing and understanding connected entities and their relations. This is in contrast with the flattened RDF representations, which were the predominant prior paradigm.

The information contained in a knowledge graph can be accessed through queries in specialized graph programming languages such as SPARQL (RDF) and Cypher (PGs) (Francis et al., 2018). However, constructing these queries to fulfill a specific information need requires significant expertise. Consequently, systems that enable querying KGs via natural language interfaces are becoming increasingly important (Gu et al., 2022). The task of semantic parsing focuses on automatically generating query programs from natural language inputs that express a particular intent.

Semantic parsing continues to be a challenging and demanding task. Typically, developing a semantic parser for a target KG requires the availability of annotated training data. Such data is often scarce and acquiring more can be expensive due to the considerable human effort required. Moreover, these training datasets are not easily transferable. Each KG defines, in principle, its own translation language, with its own schema of specific relations, concepts and properties that require domain-specific annotations. These challenges are further exacerbated when considering property graphs, which have been significantly understudied compared to RDF-style graphs.

We believe that automatic data generation is the answer to many of these challenges. In this work we introduce SPOT (zero-shot **S**emantic **P**arsing **O**ver proper**T**y graphs), a method that can automatically generate training data starting from a knowledge graph, without the need for human intervention. Our approach begins with an ungrounded tree pattern comprising a set of anonymous nodes and edges that represent a graph query. We match this pattern to the knowledge graph, producing a grounded version with instantiated entities and attributes. From this grounded tree, we derive the corresponding executable query program.

Additionally, from the grounded tree pattern, we generate a natural language realization, which we refer to as proto-NL. We employ a finite-state transducer that takes as input a linearized tree and sequentially outputs natural language fragments corresponding to nodes and relations. We then use a large language model to paraphrase the proto-NL into a final natural language realization (NL). By

Findings of the Association for Computational Linguistics: ACL 2025, pages 10057–10073

pairing the generated queries with their corresponding NLs, we can generate a training data-set that can be utilized to train a semantic parsing model.

We demonstrate the effectiveness of SPOT on property graphs using two different datasets containing queries written in Cypher. Our results show that with SPOT we can obtain a semantic parser that significantly outperforms the de-facto alternative used for zero-shot semantic parsing over PGs (i.e. prompting an LLMs with the NL query and the graph schema). Although SPOT was designed to leverage the characteristics of property graphs, we can also apply it to RDF graphs. Our experiments show that in this setting our approach outperforms BYOKG (Agarwal et al., 2024a), a state-of-the-art method for zero-shot semantic parsing over RDF graphs.

The idea of sampling some form of tree pattern directly from the KG for data generation is not novel, a similar approach has been proposed in the context of RDF graphs (Agarwal et al., 2024a). Neither is the idea of exploiting the compositional nature of a representation to automatically generate an NL realization, similar ideas have been proposed for semantic parsing over relational databases (Wang et al., 2015). Our unique contribution is to combine and extend those ideas to develop an effective data generation approach for PGs; to the best of our knowledge this is the first approach for zero-shot semantic parsing over PGs. Our code is available at `https://github.com/interact-erc/SPOT`

The paper is organized as follows: Section 2 provides some preliminaries on knowledge graphs, Section 3 describes our data generation approach, Section 4 presents our method applied to property graphs, including the experimental setup and results, Section 5 applies the same approach to RDF graphs, Section 6 describes the related work and finally Section 7 concludes the paper.

2 Preliminaries

A knowledge graph (KG) is a structured representation of entities and their relations organized as a graph where nodes represent entities and edges relations. The information stored in the KG can be efficiently retrieved by executing programs (i.e. formal queries) which can be written in different graph query languages.

A commonly used KG format popularized in the early 2000s is the Resource Description Framework (RDF). RDF graphs consist of entity-classes, entities and relations stored in triplet format. A triplet $< s, p, o >$ represents a subject s, a predicate p and an object o. The predicate p defines the relation (from s to o) that connects two entities of a certain class type. These graphs are queried via the SPARQL query language.

Property graphs (PGs) also consist of nodes and edges, but unlike RDF graphs, nodes in PGs are structured objects. In addition to belonging to a class type, they can have an arbitrary list of associated properties, each represented as a key-value pair storing specific information about the entity. In general, PGs are believed to enable the specification of a rich and semantically intuitive data model. This is because they clearly distinguish between entity properties and entity relations, whereas RDF graphs conflate the two concepts: entity properties can only be expressed as relations. Formal query languages for PGs include Cypher (Francis et al., 2018) and Gremlin (Rodriguez, 2015). Appendix D shows a fragment of the schema of one of the PGs used in our experiments.

3 Generation with Tree Patterns and Finite State Transductions

Figure 1 gives a general overview of our data generation approach which consists of five main steps:

- We generate tree patterns consistent with the graph schema.
- We ground the free variables of the tree patterns on the KG to obtain grounded tree patterns.
- We linearize the tree patterns and generate a set of sequences that represent the paths of the tree.
- We run a finite state transducer over the paths to generate an NL realization of the grounded pattern which we refer to as proto-NL. We also generate the Cypher query corresponding to the grounded tree pattern.
- Finally, we prompt an LLM to paraphrase the proto-NL to obtain the final NL realization.

In the next sub-sections we describe each of the steps in more detail.

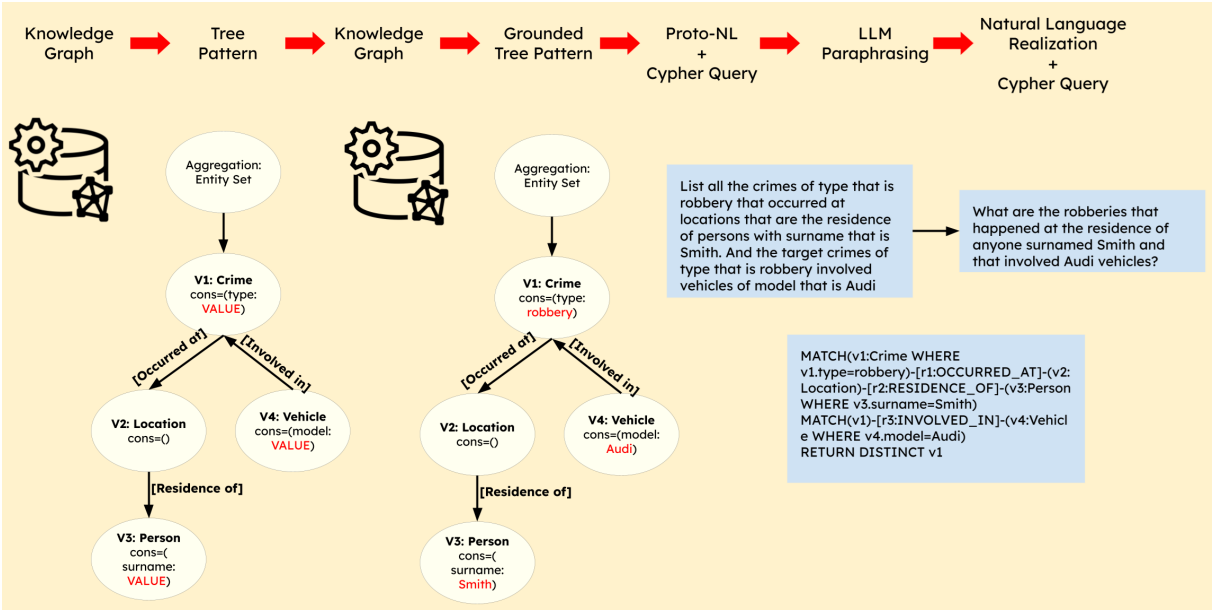

Figure 1: Overview of SPOT data generation approach

3.1 Tree Pattern Generation

A grounded tree pattern is designed to represent a canonical Cypher query consisting of a set of matching path statements. The root of the tree is a special aggregation node that has a single child that represents the answer node. More formally, a grounded tree pattern consists of a list of j nodes:

$$T = [v_0, v_1, v_2, \ldots, v_j] \quad (1)$$

The first node v_0 is a special node that represents an aggregation operation. Intuitively, an aggregation operation is a function that can be applied over a set of nodes to produce an answer to a query. We implement seven types of aggregations: set, set size, attribute value set, max attribute value set, min attribute value set, arg-max attribute value and arg-min attribute value. The second node in the tuple represents the expected answer of the query, all tree patterns are rooted at this answer node.

Each node in the tree is a tuple:

$$v =< L, A, P, R, D, C > \quad (2)$$

where L is the concept class of v, A is an optional set of pairs of attribute value constraints, P is the parent of v, R is the label of the relation that connects v to its parent, D is the direction of the relation (i.e. wether v is the domain or range of the relation) and C is a list of all the children of v in the tree pattern. Figure 1 shows an example of a grounded tree pattern of four nodes (excluding aggregation node) that would be written as:

$$T = [v_0, v_1, v_2, v_3, v_4] \quad (3)$$

with:

$$\begin{aligned} v_1 &: < crime, \{tp. : robbery\}, v_0, root, ran., \{v_2, v_3\} > \\ v_2 &: < location, \{\}, v_1, occ\_at, ran., \{v_3\} > \\ v_3 &: < person, \{sur. : Smith\}, v_2, res\_of, ran., \{\} > \\ v_4 &: < vehicle, \{model : Audi\}, n_1, inv\_in, dom., \{\} > \end{aligned} \quad (4)$$

Each grounded tree pattern can be easily mapped to its corresponding Cypher query. This is done by creating a *match* statement for every path in the tree and adding the appropriate *where* constraints on each node. For example the grounded tree pattern of the previous example will be mapped to the Cypher query:

```
MATCH (v1:Crime)-[r1:OCCURRED_AT]-(v2:
    Location)-[r2:RESIDENCE_OF]-(v3:
    Person WHERE v3.surname = 'Smith')
MATCH (v1)-[r3:INVOLVED_IN]-(v4:Vehicle
    WHERE v4.model = 'Audi')
RETURN DISTINCT v1
```

While every grounded tree pattern can be mapped to a Cypher query, the reverse is not always true, as some Cypher queries do not have an equivalent grounded tree representation. A clear example are Cypher queries that include sub-queries. This being said, the set of Cypher queries that can be represented as grounded tree patterns is quite rich and covers a good proportion of all Cypher queries. In fact, all the observed patterns in the available

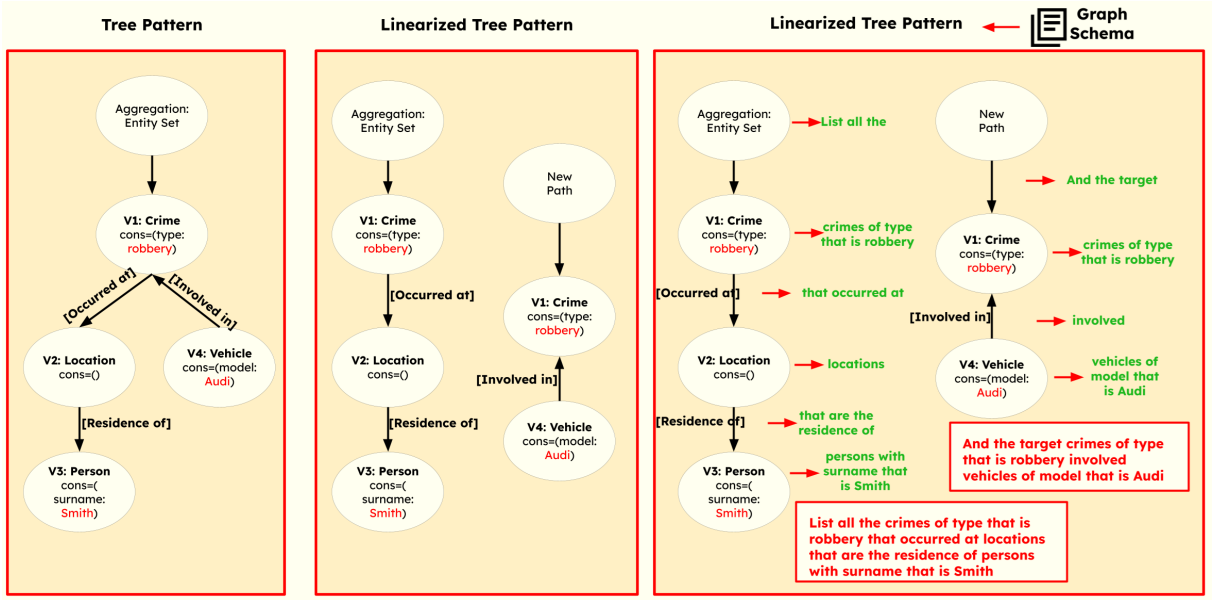

Figure 2: Illustration of how SPOT derives the proto-NL realization.

benchmarks can be expressed as grounded tree patterns. In any case, we believe generalizing the tree patterns to increase expressivity shouldn't be difficult, but we leave this as future work. This will also require augmenting current benchmarks with more complex patterns to properly test the added expressivity.

To produce the training dataset, we take a three-stage approach. In the first stage, we generate ungrounded tree patterns of fixed length n with a random generative process which uses the graph schema to ensure that the resulting pattern is valid. The recursive generative process starts by randomly creating a root aggregation node and then at each step a new node (and labeled linking edge) is attached with uniform probability from all valid descendants. The process stops when n nodes have been generated. We repeat this process for various target tree sizes (i.e. number of nodes). In the second stage we ground the tree pattern by finding an instantiation of the constraints which leads to a Cypher query with a non-empty return. We discard grounded trees with empty returns. Finally, we expand the set of grounded trees by ensuring that for each tree we also include in the training set all of its sub-trees and corresponding Cypher queries.

3.2 Proto-NL Generation

Figure 2 illustrates our approach for generating a natural language realization of a grounded tree pattern, we call this realization proto-NL. The approach involves two steps. In the first step we linearize the tree by listing all the paths of the graph in a pre-order tree traversal, this step also involves inserting an additional special "new path" node into all paths that do not start at the root.

In a second step we run a finite state transducer that takes as input the linearized representation of the tree and outputs a natural language realization of the corresponding grounded tree pattern. The finite state transducer leverages the inherent compositionality of the linearized tree pattern and is automatically built from the graph schema (see Appendix D). The idea is simple. One cannot assume that any NL realization of a grounded tree pattern can be expressed as a monotonic transduction of the linearized pattern representation. However, what we can assume is that every tree pattern has one monotonic realization. This realization might be redundant and not very fluent (which is why we refer to it as "proto" NL), but as long as it is a valid realization it will serve its purpose. All that we need is one valid realization, as we will leverage the capabilities of LLMs to paraphrase and generate a more fluid and less redundant NL.

A finite state transducer can be viewed as the simplest form of a synchronous grammar, one in which the languages are regular. It should be possible to generalize our approach to consider more general synchronous grammars, which would allow some re-orderings of the NL realization. This is however more challenging, since such a grammar needs to be automatically constructed from the graph schema alone. We leave this as a possible

direction of future work. In practice, we would like to point out that we observed that the paraphrasing step can indeed perform re-orderings.

3.3 Proto-NL Rephrasing

As it can be seen in Appendix A the proto-NL generated by the finite state transducer can be redundant and may not reflect the way humans typically use language. To address this, we apply a paraphrasing step in which we prompt an LLM to rephrase the proto-NL (see Appendix E). For PGs this simple approach already produces a fluent and "natural sounding" NL realization.

As it was already mentioned in Section 2, RDF graphs conflate the concept of relation and entity property. As a result, the proto-NLs generated from grounded tree patterns sampled from RDF graphs are more distant from a "natural sounding" NL realization. In preliminary experiments we observed that for these proto-NLs it was better to perform the paraphrasing in two steps. Recall that we linearize the tree pattern to obtain a set of paths, in the first step we paraphrase each of these paths separately. Then in a second step we prompt the LLM to combine the NLs corresponding to each path into a single sentence.

4 Property Graphs Experiments

4.1 Experimental Setup

To test our method we generate training data with SPOT and use it to train a transformer decoder-model. We generate 100k of samples and for each proto-NL we generate 3 paraphrasis (see Appendix B for a discussion). To avoid generating samples that are semantically meaningless, we do not generate queries that contain three relations of the same type, 2 or more edge nodes without defined property, nodes that repeat the same entity values, edge nodes of the same class as the answer node without a defined property and parallel nodes of the same class that have one an instantiated property and the other not. The model is trained to predict a formal query for a given natural language question. Training uses generated data only, thus it can be regarded as a zero-shot setting.

Note that to focus purely on the semantic parsing task we assume gold entity linking and we pass the gold entity values in the input.

Specifically, we use *Qwen3-4B* (Yang et al., 2025) and *Llama-3.2-3B* (Grattafiori et al., 2024) as our base parsers. For generating the natural language realization from our proto-NL we employ *Qwen3-14B*. We do an initial parameter selection by reserving a small subset of the data, and then proceed to train on the full dataset. We train the model on a single A100 GPU and every result reported is the average of 5 runs with different seeds. As evaluation metric we report execution accuracy, where a predicted query is considered correct if it returns the same results as the gold query when run on the corresponding KG.

For the PGs we focus on the Cypher graph language and evaluate on the following datasets:

- **ZOGRASCOPE** (Cazzaro et al., 2025). The benchmark is based on a Crime Investigation Graph[1] which represents relations between persons, locations and events in the context of criminal activities. The graph contains more than 60k nodes and 10k edges. The schema defines 11 entity classes, 32 unique properties and 17 relation types. In total there are 5k human annotated samples, with ~3k allocated to the training set. The benchmark comes with three partitions: i.i.d, compositional and length. We maintain the partitions distinction but note that for our approach they all fall under the zero-shot scenario. We use version 1.0 of ZOGRASCOPE.

- **DMT**. A private industrial dataset that represents academic publications, research data and other research outcomes interlinked to authors, organizations and funding agencies. The graph contains more than 13k nodes and 90k edges. The schema defines 10 entity classes 28 unique properties and 28 relation types. In total the test set contains 2k samples.

For ZOGRASCOPE we also provide results training our base parsers on the human annotated training set (while DMT does not have one).

Since, to the best of our knowledge, we are the first ones to explore zero-shot semantic parsing for PGs Cypher graphs, we will compare our method with the natural strategies for zero-shot parsing based on LLM prompting.

The zero-shot setting employs a prompt (Appendix F) that includes the description of the graph schema and then asks the LLM to provide the Cypher query for a given question. Additionally, we report results for the few-shot setting, where

[1] https://github.com/neo4j-graph-examples/pole

	Model	ZOGRASCOPE			DMT
		iid	compositional	length	
Training Set Fine-tuning	Qwen 3-4B	**98.04**	72.42	20.19	-
	Llama 3.2-3B	97.90	70.99	25.88	-
	w. SPOT *nl*	83.46	54.78	17.23	-
SPOT *Fine-tuning*	Qwen 3-4B	78.38	**77.16**	**66.56**	-
	Llama 3.2-3B	78.68	75.46	64.00	**74.36**
Few-shot	GPT-4o	66.40	63.71	22.66	-
	Mistral 7B	44.44	25.30	11.33	-
	Qwen3 4B	43.27	28.76	12.69	-
	Llama 3.2-3B	31.03	15.49	5.98	-
Zero-shot	GPT-4o	41.67	32.91	16.28	53.66
	Mistral 7B	8.85	4.82	0.56	5.07
	Qwen 3-4B	10.41	7.04	0.88	4.17
	Llama 3.2-3B	3.38	1.48	0.24	20.70

Table 1: Execution accuracy on the PG benchmarks.

5 examples are included in the prompt. For ZOGRASCOPE we use the results reported in the original paper. The evaluated models include GPT-4o (*gpt-4o-2024-08-06*), a large proprietary model, as well as small LLMs that can be run locally: *Llama-3.2-3B*, *Qwen3-4B* (Yang et al., 2025) and *Mistral-7B-v0.1* (Jiang et al., 2023).

4.2 Results

Table 1 reports results for the PG experiments. We observe that SPOT consistently outperforms the LLM prompting zero-shot alternative strategies. It obtains 77.16% and 74.36% accuracy versus 32.91% and 53.66% of the best out-of-the-box LLM. This shows that simple approaches based solely on prompting general LLMs (even large proprietary ones) do not yield satisfactory results. This is still true in the few-shot setting: while providing examples notably improves LLMs performance, it still falls short of SPOT's accuracy.

Regarding the relative performance of the different LLMs, we observe that the smaller local models exhibit very low accuracy. The bigger proprietary LLM on the other hand does perform significantly better, which is remarkable for a general model.

Compared to fine-tuning on the training set, SPOT is behind in the i.i.d. setting but achieves strong results on the other two partitions. It is important to note that these partitions are defined relative to the training set, whereas SPOT may have been exposed to similar patterns during its training phase. This reflects a key advantage of SPOT: its ability to generate a large volume of synthetic examples, in contrast to the limited availability of human-annotated data (the training set only contains ~3k samples).

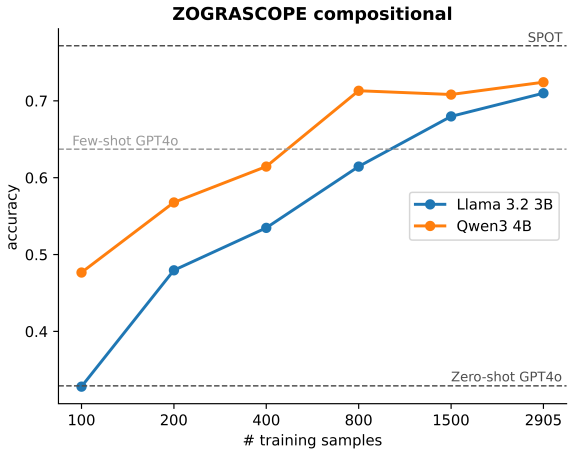

Figure 3: Learning curve for the ZOGRASCOPE benchmark training data on the compositional partition.

If we look at the third row of Table 1, we observe the performance when the human-annotated natural language sentences in the training set are replaced with NLs generated via the SPOT process. This suggests that there is still a significant margin for improving the synthetic NL realizations, which potentially is one of the key component to further improve performance.

Figure 3 shows the parser's performance during

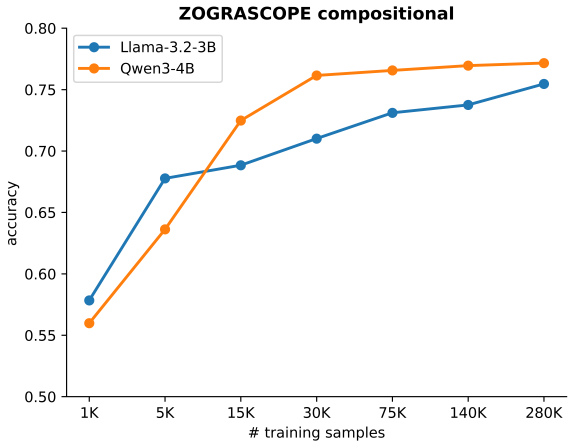

Figure 4: SPOT learning curve for the ZOGRASCOPE benchmark on the compositional partition.

fine-tuning with the human-annotated training set, as a function of the set's size. We observe that with 100 annotated samples the performance improves over zero-shot prompting. On the other hand even with the full-annotated set of ~3k samples, the performance of the supervised model is still below the performance of SPOT.

Finally, Figure 4 shows performance of SPOT as a function of the size of the generated training set used to fine-tune the semantic parser for the ZOGRASCOPE compositional partition. Interestingly, the learning curves exhibit the typical pattern of learning curves for supervised learning, namely an initial phase of exponential performance improvements followed by a phase of linear growth.

5 RDF Graphs Experiments

5.1 Experimental Setup

For RDF graphs we use the following datasets:

- **GRAILQA** (Gu et al., 2021). A large-scale knowledge base question answering dataset grounded on Freebase using the SPARQL query language. It consists of 64k questions and it comes with three partitions: i.i.d, compositional and zero-shot. We maintain the partitions distinction but note that for our approach they all fall under the zero-shot scenario. The GRAILQA test set is not public, we follow the literature in reporting results over the validation set since it has been shown that it correlates consistently with the test (Gu and Su, 2022).

- **GRAILQA++** (Dutt et al., 2023). An extension of GRAILQA designed to be more challenging by including more complex questions and query patterns. While the benchmark comes with different portions, we evaluate on the one associated to the GRAILQA dataset.

We employ as base parser *Llama-3.2-3B* and *Llama-3.1-8B* as the model for LLM paraphrasing. Note that we assume gold entity linking and we pass the gold entity values in input. Besides training with data generated by SPOT, we also train our base parser with different training data:

- **SUPERVISED**. The official training set available for GRAILQA. This is a large manually annotated dataset with 44k samples, we expect the performance obtained by fine-tuning with this data to serve as a form of upper-bound for zero-shot performance.

- **BYOKG**. Data generated with a state-of-the-art method proposed in Agarwal et al. (2024a). This is a method specifically designed for RDF graphs, BYOKG starts from a random node in the graph and through random traversal generates a query. The query is then converted into an S-expression (Su et al., 2016) which serves as the starting point for obtaining the natural language realization. This is accomplished by querying an LLM in an involved multi-step process that generates the NL for subcomponents of the S-expression, eventually leading to the final realization. We use the original code provided by the authors to generate the data with the only difference that we upgrade their LLM model to be *Llama-3.1-8B* so that it is directly comparable with ours.

Note that the zero-shot strategies that we tested for PGs cannot be applied to GRAILQA since the Freebase schema is too big to be contained in the prompt. For SPOT we generate 200k samples and we do not experiment with multiple paraphrasis. We do not generate queries that have repeated relations or entities.

5.2 Results

Table 2 reports results for the RDF graphs experiments. The first row of the table shows the performance of a model trained with fully supervised data: both the SPARQL queries and their corresponding NL realizations come from the gold training data. The second and third rows of the

Model	GRAILQA			GRAILQA++
	iid	compositional	zero-shot	
SUPERVISED	96.17	52.18	0.19	4.75
w. SPOT *nl*	54.99	36.26	0.84	4.43
w. BYOKG *nl*	50.97	31.24	0.5	4.27
SPOT	**39.50**	**39.43**	**53.31**	**39.29**
BYOKG	37.78	32.41	51.49	38.05

Table 2: Execution accuracy on the RDF graphs benchmarks.

table show the performance of a semantic parser trained with gold training SPARQL queries and automatically generated natural language realizations using either SPOT or BYOKG. To be more precise, for SPOT we take the SPARQL queries from the GRAILQA train set, we convert them to tree patterns and then run the transducer to generate the associated proto-NL which is in turn paraphrased to produce the final NL. Likewise, for BYOKG we compute the S-expression mappings of the GRAILQA SPARQL queries and generate the NL realizations using the BYOKG method. Finally, rows 4 and 5 of Table 2 show the zero-shot setting, that is training with both SPARQL queries and natural language realizations automatically generated with either SPOT or BYOKG.

As expected the performance in the iid partition of the fully supervised approach is very high. This is not surprising since the iid partition would contain many query patterns that are also observed in the training partition. However, performance drops significantly in the compositional partition, known to be challenging for transformer models (Lake and Baroni, 2018). Accuracy is also very low on both the zero-shot partition and the GRAILQA++ benchmark, overall suggesting a poor generalization of the fully supervised model.

In comparison, while the zero-shot parsers exhibit much lower performance on the iid setting, they perform much better in the zero-shot partition and on the more complex queries of the GRAILQA++ benchmark. This suggests that the automatic data generation approach could potentially produce semantic parsers with better generalization. Probably, due to the fact that automatic query generation can create a broader set of diverse queries with a greater coverage of all parts of the target graph, something that would be difficult and costly to achieve through manual annotation.

Comparing the two zero-shot data generation approaches we observe that even though SPOT was not designed with RDF graphs in mind, it consistently outperforms BYOKG across all partitions.

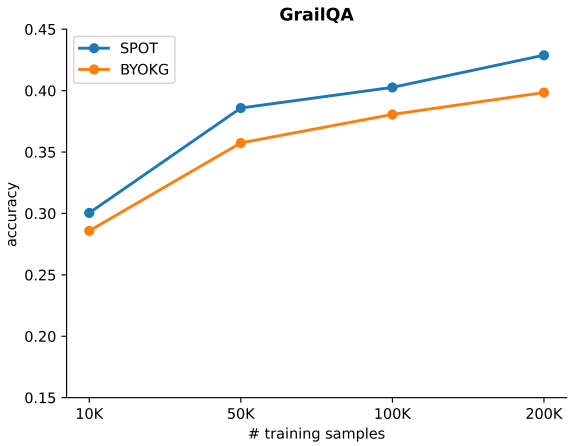

Figure 5: Learning curves for GRAILQA and GRAILQA++. The average across partitions is reported.

We now go back to Table 2 to focus on the first three rows. The first thing to point out is that our natural language realization achieves higher performance than BYOKG. We believe that employing the proto-NL as a starting point for our pipeline is an advantage of our approach. The second thing to notice is that both NL generation methods perform significantly lower than the fully supervised upper-bound on the IID partition. This suggests that there is still a significant margin for improving the NL generation for RDF graphs.

Figure 5 shows performance as a function of the size of the generated training set used to fine-tune the semantic parser for the GrailQA benchmark. Execution performance is averaged over all test partitions, for a full breakdown refer to appendix C. The maximum size shown on the figure (i.e. 200k samples) corresponds to the main results of table 2. Both models exhibit a similar trend, performance continues to improve as we increase the amount of

training data but at a slower rate.

6 Related Work

Recent years have witnessed a growing interest in natural language interfaces to knowledge graphs, focusing mainly in semantic parsing and question answering (KBQA) (Rubin and Berant, 2021; Das et al., 2021; Sima et al., 2021; Lewis et al., 2021; Gu and Su, 2022; Ye et al., 2022; Shu et al., 2022; Dutt et al., 2022; Gu et al., 2023; Yu et al., 2023; Awasthi et al., 2023; Zhou et al., 2025). Some of these works leverage in-context learning approaches (Tian et al., 2023; Tan et al., 2023a; Li et al., 2023b; Baek et al., 2023) as well as graph RAG (Peng et al., 2024). There seems to be a general consensus that LLMs do not offer satisfactory performance out of the box (Li et al., 2023a; Tan et al., 2023b; Liu et al., 2024).

Zero-shot semantic parsing over knowledge graphs is a relatively newer subject that is starting to gain the interest of the research community, but it remains a relatively unexplored subject area. In the context of zero-shot semantic parsing for RDF graphs, the work that is most related to ours is BYOKG (Agarwal et al., 2024a). BYOKG generates data from an RDF KG by a random traversal which generates S-expressions (Su et al., 2016) that are then used to generate natural language realizations via an iterative paraphrasing strategy. The authors test their data generation approach with an in-context learning strategy. Besides this zero-shot approach, there have also been some contributions focusing on few-shot semantic parsing for RDF, including Gu et al. (2023) and Agarwal et al. (2024b). In the context of knowledge bases, there have been some methods: Li et al. (2023b) and Li et al. (2024b) that generate logical forms. However, their setting is not truly zero-shot since the method uses unlabeled queries taken from an annotated training set.

Strikingly, despite the recent popularity gained by property graphs in the industrial sector, there is relatively little work on semantic parsing specifically dedicated to these types of graphs. Of the available literature there are some works that focus on fine-tuning (Guo et al., 2022; Zhao et al., 2022; Tiwari et al., 2025; Zhong et al., 2025) or on exploiting large language models (Li et al., 2024c; Liang et al., 2024a; Zhou et al., 2024; Liang et al., 2024b). There have also been some efforts on mapping RDF graphs to Cypher (Moreira and Ramalho, 2020; A Agrawal et al., 2022; Nie et al., 2022; Feng et al., 2024). Besides these prior literature on PGs, to the best of our knowledge this is the first work specifically dedicated to zero-shot semantic parsing over PGs.

7 Conclusions

Natural language interfaces to knowledge graphs and databases have the potential to democratize data-science by enabling people with no programming experience to efficiently and precisely interact with data. The main blocker preventing the development of such interfaces is the training data bottleneck when fine-tuning semantic parsers. In this paper we addressed this bottleneck in the specific context of semantic parsing for property graphs. We presented a data-generation method that leverages the inherent compositional nature of graph queries. One key observation is that although the mappings between queries and their natural language realizations can be very complex, there might still exist one "simpler" natural language realization that can be automatically generated as a monotonic transduction of an intermediate tree pattern representation. Then we can use LLM paraphrasing to generate the more "complex" realizations. Our experiments show the potential of the proposed approach and bring us a step closer in developing zero-shot semantic parsers that have actual practical use as industrial applications.

Limitations

There are several limitations with our current approach to zero-shot semantic parsing for PGs:

- As already pointed out in the paper not all Cypher queries can be mapped to the intermediate tree pattern representation. For example, our intermediate tree representation does not consider sub-queries which are quite important. In the future we plan to extend the intermediate representation to include sub-queries. Also, and this is probably more important, we would like to create more complex benchmarks for semantic parsing over PGs that include such queries.

- Our tree pattern generation strategy leads to queries that are syntactically correct but obviously a syntactically correct query doesn't imply a semantically meaningful query, i.e. a

query that makes sense. This is not necessarily a problem if the semantic parser can still learn from such 'noisy' queries. Still, we believe that it is important to explore ways in which the queries could be filtered.

- Although generating training data with our approach is cheap, fine-tuning a model is not, especially for very large LMs. So in the future we plan to work on generation strategies with a focus on maximizing the diversity of the generated set.

- Our approach relies on the ability of an LLM to provide good paraphrases, but it is possible that for some complex patterns the paraphrasing would fail. We believe some additional post-processing of the proto-NL might be of help, for example we could explore alternative ways for handling co-references in the generation.

- Ultimately we don't believe that a semantic parser can be trained to perfection with generated data alone. Eventually, there will always be some complex cases for which we will need to elicit some form of human supervision. But what we want is to limit annotations to complex cases only, so future work should combine the data creation approach with an active learning strategy.

- Our approach is zero-shot because it requires no-training data but it does require training time. This means that an end user could not just load a PG and immediately interact with it. Speeding this initial training will also be part of the challenge to make semantic parsing over PGs truly useful.

- With our approach it is possible to generate arbitrary complex tree patterns (i.e. involving many nodes and relations) and find a corresponding Cypher query and even a natural language realization. But chances are that in a real interaction with a PGs the end user won't express his information need with a single complex query. Instead, the expected interaction would be a dialogue with the system in which the queries get iteratively refined. We believe one of the most exciting paths of future work is to extend our approach to generate data to train not a one-round semantic parsing interaction, but rather a dialogue consisting of multiple rounds of semantic parsing.

- We would still like our approach to work better with RDF graphs since they are also an important type of representation. One possibility for this would be to develop automatic ways of transforming RDF graphs so as to "un-flatten" their structure and make them look more like property graphs.

Risks and Ethical Considerations

We believe our method contributes to enabling easier access to information in graphs. We do not consider our work to present risks or ethical concerns. However, we recognize that systems leveraging automatically generated data should be implemented cautiously and with human supervision to mitigate the risk of reinforcing factual errors or biases present in the KGs.

Acknowledgments

We thank Ritam Dutt and Dongfang Ling for providing us with access to GRAILQA++.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 853459. The authors gratefully acknowledge the computer resources at ARTEMISA, funded by the European Union ERDF and Comunitat Valenciana as well as the technical support provided by the Instituto de Física Corpuscular, IFIC (CSIC-UV). This research is supported by a recognition 2021SGR-Cat (01266 LQMC) from AGAUR (Generalitat de Catalunya).

References

Lakshya A Agrawal, Nikunj Singhal, and Raghava Mutharaju. 2022. A sparql to cypher transpiler: Proposal and initial results. In *Proceedings of the 5th Joint International Conference on Data Science & Management of Data (9th ACM IKDD CODS and 27th COMAD)*, CODS-COMAD '22, page 312–313, New York, NY, USA. Association for Computing Machinery.

Bilal Abu-Salih. 2021. Domain-specific knowledge graphs: A survey. *Journal of Network and Computer Applications*, 185:103076.

Dhruv Agarwal, Rajarshi Das, Sopan Khosla, and Rashmi Gangadharaiah. 2024a. Bring your own KG: Self-supervised program synthesis for zero-shot

KGQA. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 896–919, Mexico City, Mexico. Association for Computational Linguistics.

Prerna Agarwal, Nishant Kumar, and Srikanta Bedathur. 2024b. SymKGQA: Few-shot knowledge graph question answering via symbolic program generation and execution. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10119–10140, Bangkok, Thailand. Association for Computational Linguistics.

Abhijeet Awasthi, Soumen Chakrabarti, and Sunita Sarawagi. 2023. Structured case-based reasoning for inference-time adaptation of text-to-sql parsers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(11):12536–12544.

Jinheon Baek, Alham Fikri Aji, and Amir Saffari. 2023. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering. In *Proceedings of the 1st Workshop on Natural Language Reasoning and Structured Explanations (NLRSE)*, pages 78–106, Toronto, Canada. Association for Computational Linguistics.

Francesco Cazzaro, Justin Kleindienst, Sofia Marquez, and Ariadna Quattoni. 2025. Zograscope: A new benchmark for property graphs. *Preprint*, arXiv:2503.05268.

Rajarshi Das, Manzil Zaheer, Dung Thai, Ameya Godbole, Ethan Perez, Jay Yoon Lee, Lizhen Tan, Lazaros Polymenakos, and Andrew McCallum. 2021. Case-based reasoning for natural language queries over knowledge bases. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9594–9611, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ritam Dutt, Kasturi Bhattacharjee, Rashmi Gangadharaiah, Dan Roth, and Carolyn Rose. 2022. PerKGQA: Question answering over personalized knowledge graphs. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 253–268, Seattle, United States. Association for Computational Linguistics.

Ritam Dutt, Sopan Khosla, Vinayshekhar Bannihatti Kumar, and Rashmi Gangadharaiah. 2023. GrailQA++: A challenging zero-shot benchmark for knowledge base question answering. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 897–909, Nusa Dua, Bali. Association for Computational Linguistics.

Yanlin Feng, Simone Papicchio, and Sajjadur Rahman. 2024. Cypherbench: Towards precise retrieval over full-scale modern knowledge graphs in the llm era. *arXiv preprint arXiv:2412.18702*.

Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1433–1445, New York, NY, USA. Association for Computing Machinery.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Yu Gu, Xiang Deng, and Yu Su. 2023. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4928–4949, Toronto, Canada. Association for Computational Linguistics.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond i.i.d.: Three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, WWW '21, page 3477–3488, New York, NY, USA. Association for Computing Machinery.

Yu Gu, Vardaan Pahuja, Gong Cheng, and Yu Su. 2022. Knowledge base question answering: A semantic parsing perspective. In *4th Conference on Automated Knowledge Base Construction*.

Yu Gu and Yu Su. 2022. ArcaneQA: Dynamic program induction and contextualized encoding for knowledge base question answering. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 1718–1731, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Aibo Guo, Xinyi Li, Guanchen Xiao, Zhen Tan, and Xiang Zhao. 2022. Spcql: A semantic parsing dataset for converting natural language into cypher. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, page 3973–3977, New York, NY, USA. Association for Computing Machinery.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills

of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2873–2882. PMLR.

Patrick Lewis, Yuxiang Wu, Linqing Liu, Pasquale Minervini, Heinrich Küttler, Aleksandra Piktus, Pontus Stenetorp, and Sebastian Riedel. 2021. Paq: 65 million probably-asked questions and what you can do with them. *Transactions of the Association for Computational Linguistics*, 9:1098–1115.

Harry Li, Gabriel Appleby, Camelia Daniela Brumar, Remco Chang, and Ashley Suh. 2024a. Knowledge graphs in practice: Characterizing their users, challenges, and visualization opportunities. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):584–594.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Ma Chenhao, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023a. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems*, volume 36, pages 42330–42357. Curran Associates, Inc.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023b. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6966–6980, Toronto, Canada. Association for Computational Linguistics.

Zhenyu Li, Sunqi Fan, Yu Gu, Xiuxing Li, Zhichao Duan, Bowen Dong, Ning Liu, and Jianyong Wang. 2024b. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering. *Preprint*, arXiv:2308.12060.

Zhuoyang Li, Liran Deng, Hui Liu, Qiaoqiao Liu, and Junzhao Du. 2024c. Unioqa: A unified framework for knowledge graph question answering with large language models. *Preprint*, arXiv:2406.02110.

Yuanyuan Liang, Keren Tan, Tingyu Xie, Wenbiao Tao, Siyuan Wang, Yunshi Lan, and Weining Qian. 2024a. Aligning large language models to a domain-specific graph database for nl2gql. *Preprint*, arXiv:2402.16567.

Yuanyuan Liang, Tingyu Xie, Gan Peng, Zihao Huang, Yunshi Lan, and Weining Qian. 2024b. Nat-nl2gql: A novel multi-agent framework for translating natural language to graph query language. *Preprint*, arXiv:2412.10434.

Jinxin Liu, Shulin Cao, Jiaxin Shi, Tingjian Zhang, Lunyiu Nie, Linmei Hu, Lei Hou, and Juanzi Li. 2024. How proficient are large language models in formal languages? an in-depth insight for knowledge base question answering. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 792–815, Bangkok, Thailand. Association for Computational Linguistics.

Ezequiel José Veloso Ferreira Moreira and José Carlos Ramalho. 2020. SPARQLing Neo4J. In *9th Symposium on Languages, Applications and Technologies (SLATE 2020)*, volume 83 of *Open Access Series in Informatics (OASIcs)*, pages 17:1–17:10, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Lunyiu Nie, Shulin Cao, Jiaxin Shi, Jiuding Sun, Qi Tian, Lei Hou, Juanzi Li, and Jidong Zhai. 2022. GraphQ IR: Unifying the semantic parsing of graph query languages with one intermediate representation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5848–5865, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *Preprint*, arXiv:2408.08921.

Marko A. Rodriguez. 2015. The gremlin graph traversal machine and language (invited talk). In *Proceedings of the 15th Symposium on Database Programming Languages*, DBPL 2015, page 1–10, New York, NY, USA. Association for Computing Machinery.

Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive bottom-up semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 311–324, Online. Association for Computational Linguistics.

Irina Saparina and Mirella Lapata. 2024. Improving generalization in semantic parsing by increasing natural language variation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1178–1193, St. Julian's, Malta. Association for Computational Linguistics.

Yiheng Shu, Zhiwei Yu, Yuhan Li, Börje Karlsson, Tingting Ma, Yuzhong Qu, and Chin-Yew Lin. 2022. TIARA: Multi-grained retrieval for robust question answering over large knowledge base. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8108–8121, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Ana Claudia Sima, Tarcisio Mendes de Farias, Maria Anisimova, Christophe Dessimoz, Marc Robinson-Rechavi, Erich Zbinden, and Kurt Stockinger. 2021. Bio-soda: Enabling natural language question answering over knowledge graphs without training data. In *Proceedings of the 33rd International Conference on Scientific and Statistical Database Management*, SSDBM '21, page 61–72, New York, NY, USA. Association for Computing Machinery.

Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gür, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for QA evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 562–572, Austin, Texas. Association for Computational Linguistics.

Chuanyuan Tan, Yuehe Chen, Wenbiao Shao, and Wenliang Chen. 2023a. Make a choice! knowledge base question answering with in-context learning. *Preprint*, arXiv:2305.13972.

Yiming Tan, Dehai Min, Yu Li, Wenbo Li, Nan Hu, Yongrui Chen, and Guilin Qi. 2023b. Can chatgpt replace traditional kbqa models? an in-depth analysis of the question answering performance of the gpt llm family. *Preprint*, arXiv:2303.07992.

Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V. Chawla, and Panpan Xu. 2023. Graph neural prompting with large language models. *Preprint*, arXiv:2309.15427.

Aman Tiwari, Shiva Krishna Reddy Malay, Vikas Yadav, Masoud Hashemi, and Sathwik Tejaswi Madhusudhan. 2025. Auto-cypher: Improving LLMs on cypher generation via LLM-supervised generation-verification framework. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 623–640, Albuquerque, New Mexico. Association for Computational Linguistics.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. 2022. RNG-KBQA: Generation augmented iterative ranking for knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6032–6043, Dublin, Ireland. Association for Computational Linguistics.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Yang Wang, Zhiguo Wang, and Bing Xiang. 2023. DecAF: Joint decoding of answers and logical forms for question answering over knowledge bases. In *The Eleventh International Conference on Learning Representations*.

Ziyu Zhao, Michael Stewart, Wei Liu, Tim French, and Melinda Hodkiewicz. 2022. Natural language query for technical knowledge graph navigation. In *Data Mining*, pages 176–191, Singapore. Springer Nature Singapore.

Zijie Zhong, Linqing Zhong, Zhaoze Sun, Qingyun Jin, Zengchang Qin, and Xiaofan Zhang. 2025. SyntheT2C: Generating synthetic data for fine-tuning large language models on the Text2Cypher task. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 672–692, Abu Dhabi, UAE. Association for Computational Linguistics.

Jincheng Zhou, Yucheng Zhang, Jianfei Gao, Yangze Zhou, and Bruno Ribeiro. 2025. Double equivariance for inductive link prediction for both new nodes and new relation types. *Preprint*, arXiv:2302.01313.

Yuhang Zhou, Yu He, Siyu Tian, Yuchen Ni, Zhangyue Yin, Xiang Liu, Chuanjun Ji, Sen Liu, Xipeng Qiu, Guangnan Ye, and Hongfeng Chai. 2024. r^3-NL2GQL: A model coordination and knowledge graph alignment approach for NL2GQL. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13679–13692, Miami, Florida, USA. Association for Computational Linguistics.

A Generations

Table 3 showcases some examples of data generated by SPOT.

B Multiple Paraphrasis Ablation

In Table 4 we show the performance of SPOT when we employ 3 different paraphrasis for each generated sample versus when employing just one. We observe that multiple paraphrasis can bring noticeable improvements. This is consistent with other findings about the importance of paraphrasing such as Saparina and Lapata (2024).

C Learning Curves Breakdown

In Table 5 we offer a full results breakdown of the learning curves showed in section 5.2.

D Schema Snapshot

In this work we assume, as is often the case in practice, that knowledge graphs come with natural language descriptions of classes and relations. Here we show a partial snapshot of the schema that accompanies a knowledge graph, ZOGRASCOPE in this case. For each class, property and relation there is a description field that we use

Proto-NL	List all People named Andrea that have a family relation with any People with last name Hanson and the target People named Andrea must knows the phone number of any People with last name Kelly
NL	*List all Andrea who are in a family relation with any Hanson and know the phone number of any Kelly*

Query	`MATCH (v_1:Person WHERE v_1.name="Andrea")-[r_2:FAMILY_REL]-(v_2: Person WHERE v_2.surname="Hanson")`
	`MATCH (v_1:Person WHERE v_1.name="Andrea")-[r_3:KNOWS_PHONE]-(v_3: Person WHERE v_3.surname="Kelly")`
	`RETURN DISTINCT v_1`

Proto-NL	List all the national health service numbers of People that are in a family relation with any People named Anne that are in a family relation with any People with last name Kennedy
NL	*List all NHS numbers of people related to Anne, who are related to people with the last name Kennedy*
Query	`MATCH (v_1:Person)-[r_2:FAMILY_REL]-(v_2:Person WHERE v_2.name=" Anne")-[r_3:FAMILY_REL]-(v_3:Person WHERE v_3.surname="Kennedy ")` `RETURN DISTINCT v_1.nhs_no`
Proto-NL	What are the minimum call durations of Phone calls that were received at any Phones belonging to any People with last name Campbell
NL	*What is the minimum call duration of phone calls received by people with the last name Campbell?*

Query	`MATCH (v_1:PhoneCall)-[r_2:CALLED]-(v_2:Phone)-[r_3:HAS_PHONE]-(v_3 :Person WHERE v_3.surname="Campbell")`
	`RETURN v_1.call_duration`
	`ORDER BY toFloat(v_1.call_duration) ASC`
	`LIMIT 1`

Proto-NL	How many Crimes with type of offence equal to Vehicle crime that involved any Vehicles with model 4Runner and the target Crimes with type of offence equal to Vehicle crime must occurred at any Addresses with address 116 China Lane
NL	*How many vehicle crimes involving a 4Runner occurred at 116 China Lane?*

Query	`MATCH (v_1:Crime WHERE v_1.type="Vehicle crime")-[r_2:INVOLVED_IN ]-(v_2:Vehicle WHERE v_2.model="4Runner")`
	`MATCH (v_1:Crime WHERE v_1.type="Vehicle crime")-[r_3:OCCURRED_AT ]-(v_3:Location WHERE v_3.address="116 China Lane")`
	`RETURN COUNT(DISTINCT v_1)`

Proto-NL	list all tv channel that ceased operating on date which is less or equal than 1998
NL	*Which tv channel ceased operating before 1998?*

Query	`SELECT DISTINCT ?x0`
	`WHERE {`
	`?x0 :broadcast.tv_channel.to ?x1 .`
	`?x0 a :broadcast.tv_channel .`
	`FILTER (?x1 <= "1998"^^xsd:gYear)`
	`}`

Table 3: Examples of SPOT generations.

	Model	ZOGRASCOPE		
		iid	**compositional**	**length**
1 Paraphrasis	Qwen 3-4B	74.73	73.53	64.08
	Llama 3.2-3B	69.87	69.58	64.16
3 Paraphrasis	Qwen 3-4B	78.38	**77.16**	**66.56**
	Llama 3.2-3B	**78.68**	75.46	64.00

Table 4: Results of SPOT with 1 versus 3 paraphrasis per generated sample.

Model	GRAILQA			GRAILQA++
	iid	**compositional**	**zero-shot**	
BYOKG 10k	24.98	21.92	40.39	25.94
BYOKG 50k	33.18	30.00	46.40	33.38
BYOKG 100k	35.50	30.51	49.31	36.91
BYOKG 200k	37.78	32.41	51.49	38.05
SPOT 10k	28.18	23.71	42.34	25.94
SPOT 50k	36.57	34.30	50.32	33.14
SPOT 100k	37.41	34.81	51.88	36.96
SPOT 200k	39.50	39.43	53.31	39.29

Table 5: Execution accuracy of SPOT and BYOKG with training set of different sizes.

for our proto-NL generation. In case of a RDF graph the schema has the same structure with the difference that there is no properties section. Note that for the GRAILQA schema some descriptions were too wordy, so we automatically processed them with an LLM to get a shorter description fit for the proto-NL. For example the relation "*medicine.manufactured_drug_form.patent_expiry_date*" came with the associated description "*Date on which the last patent(s) protecting this drug/drug form expire*" that was automatically transformed by the LLM to be "*that will expire on*".

```
"classes": {
        "Person": {
            "description": "People"
        },
        .
        .
        .
"properties": {
        "nhs_no": {
            "description": "national health service number",
            "type": "STRING"
        },
        .
        .
        .
"relations": {
        "CURRENT_ADDRESS": {
            "description": "that lives in",
            "domain": "Person",
            "range": "Location",
            "reverse_description": "that is the residence of"
        },
        .
        .
        .
```

E Proto-NL Prompt

```
# Task
Here is a list of sentences and 3 paraphrasis for each. Words are changed and
synonyms are used creatively, while maintaining the original meaning accurately.
 The exact attribute or property requested is specified. Each piece of
information, when there are more than one, is included. All relational meanings
are preserved.

# Sentence: list the participants of Meeting with the minimum duration that were
 held in Room with Location Headquarter that hosted a Meeting with date
05/06/2012
AND the target Meeting discussed Argument that is company growth
# Paraphrasis: 1) Who were the participants in the shortest meeting about
company growth held in a room at the headquarter that also hosted a meeting on
the 5th of June 2012
2) Who were the participants involved in the shortest company growth meeting,
which took place in the same room at the headquarters that hosted another
meeting on June 5, 2012?
3) Can you identify the participants of the briefest company growth meeting held
 in a room at the headquarters, which also hosted a meeting on June 5th, 2012?
[END]

# Sentence: [[QUERY]]
# Paraphrasis:
```

F Zero-shot Baseline Prompt

```
###
Cypher schema:

CLASS: description
Person: People
Location: Locations
Phone: Phone
Email: Email
Officer: Officers
PostCode: PostCode
Area: Areas
PhoneCall: Phone calls
```

```
Crime: Crimes
Object: Criminal Objects
Vehicle: Vehicles

PROPERTY: description
year: year
postcode: post code
call_time: time
nhs_no: national health service number
address: address
name: name
phoneNo: phone number
model: model
badge_no: badge number
areaCode: area code
rank: rank
type: type
call_date: call date
call_duration: call duration (seconds)
email_address: email address
make: car brand
date: date
surname: last name
code: code
last_outcome: processing status
age: age

RELATION: description - domain -> range
CURRENT_ADDRESS: that lives in - Person -> Location
HAS_PHONE: which has - Person -> Phone
HAS_EMAIL: which has - Person -> Email
KNOWS_SN: that is friends with - Person -> Person
KNOWS: who knows - Person -> Person
HAS_POSTCODE: which has - Location -> PostCode
POSTCODE_IN_AREA: that is in - PostCode -> Area
INVOLVED_IN: that is involved in - Vehicle -> Crime
CALLER: that were made to - PhoneCall -> Phone
CALLED: that were received a - PhoneCall -> Phone
KNOWS_PHONE: knows the phone of - Person -> Person
OCCURRED_AT: that occurred at - Crime -> Location
INVESTIGATED_BY: that is investigated by - Crime -> Officer
PARTY_TO: which is involved in - Person -> Crime
FAMILY_REL: that has a family relation with - Person -> Person
KNOWS_LW: that lives with - Person -> Person
LOCATION_IN_AREA: that is included - Location -> Area
###

# Task
Give me the cypher neo4j query for the following question. (Answer with the
query only, do not add anything else).

# Sample
Question: [[QUERY]]

Query:
```