GraphQ IR: Unifying Semantic Parsing of Graph Query Languages with Intermediate Representation

Anonymous ACL submission

Abstract

Subject to the semantic gap lying between natural and formal language, neural semantic parsing is typically bottlenecked by the paucity and imbalance of data. In this paper, we propose a unified intermediate representation (IR) for graph query languages, namely GraphQ IR. With the IR’s natural-language-like representation that bridges the semantic gap and its formally defined syntax that maintains the graph structure, neural semantic parser can more effectively convert user queries into our GraphQ IR, which can be later automatically compiled into different downstream graph query languages. Extensive experiments show that our approach can consistently achieve state-of-the-art performance on benchmarks KQA PRO, OVERNIGHT and METAQA. Evaluations under compositional generalization and few-shot learning settings also validate the promising generalization ability of GraphQ IR with at most 11% accuracy improvement.

1 Introduction

By mapping natural language utterances to logical forms, the task of semantic parsing has been widely explored in various applications like querying database (Zhong et al., 2017; Yu et al., 2018) or knowledge base (Zhang et al., 2018; Talmor and Berant, 2018), virtual assistant conversation (Campagna et al., 2019; Fischer et al., 2021) and general-purpose code generation (Ling et al., 2016; Yin and Neubig, 2017; Nan et al., 2020). Early attempts in this field usually rely on compositional grammar such as CCG and DCS (Zettlemoyer and Collins, 2005; Kwiatkowski et al., 2010; Liang et al., 2013). Recently, most works formulate semantic parsing as a Seq2Seq problem and adopt encoder-decoder neural networks that only require parallel data of natural language utterance and corresponding logical form for supervision (Zhong et al., 2017; Yu et al., 2020; Damonte and Monti, 2021).

However, these neural approaches still suffer from two major challenges: (a) Semantic gap. As shown in Figure 1, in graph query languages (e.g., SPARQL, Cypher, and newly emerged KoPL, etc.), graph nodes and edges constitute the key semantics of the logical forms (Pérez et al., 2009), which are very different from the expression of natural language queries. Such discrepancy significantly hinders the learning of neural semantic parser, especially in the era of pretrained language models (Lewis et al., 2020). (b) Imbalance of data. Due to the intensive labor and language-specific expertise required in annotation (Li et al., 2020), in spite of the various datasets prepared in SPARQL (Talmor and Berant, 2018; Dubey et al., 2019; Keysers et al., 2019), very few works target the semantic parsing of other graph query languages, such as Cypher and Gremlin, that are commonly used in industries (Seifer et al., 2019). Moreover, datasets of different languages are also isolated since no existing tools can support the conversion (Agrawal et al., 2022). Such imbalance and isolation of data has impeded the semantic parsing of low-resource languages in both academia and industry.

To overcome the above challenges, many works adopt complementary forms of supervision, such as the schema of database (Hwang et al., 2019), execution results of the logical forms (Clarke et al., 2010; Wang et al., 2018, 2021), constrained decoding algorithms (Shin et al., 2021; Marion et al., 2021) and canonical utterances (Berant and Liang, 2014; Su and Yan, 2017; Yu et al., 2020). Despite effective, these approaches either incur training inefficiency or performance loss (Cao et al., 2019). Besides, all these methods are tightly coupled to a specific dataset or logical form, thus cannot be easily adapted to other tasks or languages (Kamath and Das, 2018).

In this paper, we propose a unified intermediate representation for graph query languages, namely GraphQ IR, to resolve these issues from a novel
Figure 1: An example of a property graph extracted from Wikidata (Vrandečić and Krötzsch, 2014). We present a relevant user query with its corresponding logical forms in different languages and GraphQ IR.

Figure 1 demonstrates, a property graph normally includes Entity (the node types, e.g., film), Relationship (the graph edges, e.g., spouse) and Qualifier (the edge properties, e.g., start time).

with our source-to-source compiler that supports bidirectional translation between GraphQ IR and various graph query languages, our work also provides a novel toolkit that can be utilized as a transpiler for unifying different datasets or as a natural language user interface for querying graph databases.

To validate the effectiveness of GraphQ IR, we conducted extensive experiments on benchmarks KQA PRO, OVERNIGHT and MetaQA. Results show that our approach can consistently outperforms the previous works by a significant margin.

Especially under the compositional generalization and few-shot learning settings, our approach with GraphQ IR can demonstrate at most 11% increase on accuracy over the baselines with strong generalization abilities. Supplementary analysis further shows that GraphQ IR is easy to debug with 89% of the errors can be fixed with simple corrections.

The main contributions of our work include:

- We propose GraphQ IR for unifying the semantic parsing of graph query languages, and present the IR design principles that are critical to the success of semantic parsing;
- Experimental results show that our approach can achieve state-of-the-art performance across benchmarks and strong robustness even under the compositional generalization and few-shot learning settings.
- Our implemented compiler can be also utilized as a translator among different graph query languages. We will release our code and toolkit for the uses of the community.

2 GraphQ IR

We propose GraphQ IR as a novel intermediate representation that aims to bridge the semantic gap between natural and formal language as well as unify different languages to break the data bottleneck. In this section, we define the property graph, formalize the GraphQ IR, and summarize the key principles in designing GraphQ IR.

2.1 Definition

As the top of Figure 1 demonstrates, a property graph normally includes Entity (the graph nodes, e.g., Stanley Kubrick), Attribute (the node properties, e.g., date of birth), Concept (the node types, e.g., film), Relationship (the graph edges, e.g., spouse) and Qualifier (the edge properties, e.g., start time).
To exemplify, in Figure 1, the production of GraphQ IR in ex-
tended Backus–Naur form (Parr, 2013) and de-
scribed a subset of GraphQ IR’s grammar in Table 1.

| Non-terminal          | Productions                                                                 | Description                                                                 |
|-----------------------|----------------------------------------------------------------------------|----------------------------------------------------------------------------|
| S                     | → EntityQuery | AttributeQuery | RelationQuery | QualifierQuery | ValueQuery | CountQuery                                      | Start symbol of a query sequence |
| EntityQuery           | → what is EntitySet                                                        | Query that returns certain entities                                          |
| RelationQuery         | → what is the relation from EntitySet to EntitySet                         | Query that returns the relationship between given entities                  |
| CountQuery            | → how many EntitySet                                                       | Query that returns the quantity of certain entities                         |
| EntitySet             | → <ES> EntitySet LOP EntitySet </ES> | Constraint </ES> | Concept Entity |                                                        | A non-empty set of entities that match certain conditions |
| Constraint            | → whose Attribute COP Value that Relation DIR to EntitySet that Relation DIR to COP ValueEntitySet | Clauses that constrain the entities condition                                |
| Entity                | → | Concept concept | <C> |                                                        | Entity node of a specified name                                           |
| Concept               | → <C> concept |                                                        | Set of entities of a specified type                                           |
| Attribute             | → <A> attribute |                                                        | Specified property of an entity                                              |
| Relation              | → | Relation relation | </R> |                                                        | Relationship between entities                                              |
| Value                 | → VTYPE | Value | </V> | Attribute of Entity |                                                        | Specified value of certain constraints                                    |
| LOP                   | → and | or | not |                                                        | Logical operators that perform set union, intersection or exclusion         |
| COP                   | → is | { not | larger than | smaller than | at least | at most } |                                                        | Comparison operators that take value as operand                            |
| DIR                   | → forward | backward |                                                        | Relationship direction constraint                                           |
| VTYPE                 | → string | number | year | date | time |                                                        | Value type constraint                                                     |

Table 1: A subset of GraphQ IR grammar rules. We separate alternative productions at the same level using “|” and mark the terminal nodes in italics. A more complete set of grammar is presented in Appendix Table 7.

Accordingly, GraphQ IR is built on top of the above key structures as terminal nodes meantime having its productions consistent with the compositional semantics of natural language. We formally define the context-free grammar of GraphQ IR by a quadruple \( G = (V, T, S, P) \) composed of non-terminal symbols \( V \), terminal symbols \( T \), the start symbol \( S \) and production rules \( P \).

We specify the productions of GraphQ IR in extended Backus–Naur form (Parr, 2013) and describe a subset of GraphQ IR’s grammar in Table 1. To exemplify, in Figure 1, the production of GraphQ IR’s EntitySet is equivalent to a relative clause involving the operations over graph elements Concept, Relation and Entity.

2.2 Principles

We summarize several principles in designing GraphQ IR, concluded as: presenting the semantics close to the natural language while preserving the structure identical to the formal language.

2.2.1 Diminishing structural discrepancy

To facilitate the training of neural semantic parser, the target IR sequence should share similar structures in correspondence to the input utterance.

To achieve this, first, the structure of IR should match how users typically raise queries. Therefore, we simplify the triple-based structure in graph query languages into a more natural subject-verb-object syntactic construction (Tomlin, 2014).

Take Figure 1’s task setting as an example, the two triples \((?e instance_of ?c)\) and \((?e name "film")\) as the entity concept constraint in SPARQL are simplified to the sentence subject “\(<C> film </C>\)” in GraphQ IR. Multi-hop relationship and attribute queries are also formulated as relative clauses following the English expression thus can be comfortably generated by a language-model-based neural semantic parser.

Secondly, IR should also leave out the variables (e.g., \(?e, ?c\) in SPARQL) and operators (e.g., MATCH, DISTINCT, WHERE, RETURN, etc.) in graph query languages that cannot be easily aligned to natural language. Alternatively, human-readable operators are adopted in GraphQ IR for better comprehension by the language models.

2.2.2 Eliminating semantic redundancy and ambiguity

In formal languages it is quite common that multiple parallel implementations may achieve the same functionalities. However, such redundancy and ambiguity in semantics may pose challenges to the neural semantic parser.

For instance, in \(\lambda\)-DCS, there co-exists at least three implementations for constraining the entities’ concept, respectively through

- **EventNP**: \(\text{(call @getProperty (NP) (call @getProperty (call @getProperty (NP) (call @reverse (RelNP)))))}\);
- **TypeNP**: \(\text{(call @getProperty (call @singleton (TypeNP)) (string)}\)

3
Figure 2: Overall implementation of our proposed framework. The user queries are first converted to GraphQ IR sequences and subsequently translated into the downstream graph query languages by the compiler.
(Lewis et al., 2020). Therefore, BART is proficient in comprehending the diverse user utterances and generating the GraphQ IR that has natural-language-like semantic representations.

Please note that the implementation in this part is orthogonal to our GraphQ IR, thus can be also substituted by other semantic parsing models.

3.2 Compiler

The implementation of compiler comprises a front-end module that generates abstract syntax trees from GraphQ IR and a back-end module that transforms the tree structure into the target code.

Following the grammar rules formally defined in Section 2.1, the compiler front-end first performs lexical analysis and syntax analysis on the IR sequence generated in Section 3.1. The LL(*) parser that performs the leftmost derivation of a sentence with tokens look-ahead (Parr and Fisher, 2011) will automatically parse the sequence of GraphQ IR into an abstract syntax tree that contains the dependencies and hierarchical structure. The compiler back-end will then traverse the abstract syntax tree and restructure the elements and dependencies into one of the downstream graph query languages (i.e., SPARQL, KoPL, λ-DCS, Cypher, etc) following our pre-defined transformation rules. To illustrate this process, we present 2 examples of generating λ-DCS and SPARQL queries respectively in Appendix Figure 3 and Figure 4.

4 Experiments

This section evaluates the effectiveness of GraphQ IR on several benchmarks under different task settings.

4.1 Datasets

For evaluation, we choose SPARQL, KoPL, λ-DCS, and Cypher as the target graph query languages, and take KQA PRO, OVERNIGHT, METAQA as the corresponding datasets.

KQA PRO  KQA PRO (Cao et al., 2020b) is a large-scale dataset for complex question answering over Wikidata knowledge base (Vrandečić and Krötzsch, 2014). It is so far the largest KBQA corpus that contains 117,790 natural language questions along with the corresponded SPARQL and KoPL logical forms, covering diverse question types, e.g., multi-hop inference, logical union and intersection, etc. In our experiment, it is divided into 94,376 train, 11,797 dev and 11,797 test cases.

OVERNIGHT  OVERNIGHT (Wang et al., 2015) is a semantic parsing dataset with 13,682 examples across 8 different domains extracted from Freebase (Bollacker et al., 2008). Each domain has natural language questions and pairwise λ-DCS queries executable on SEMPRE (Berant et al., 2013). It exhibits diverse linguistic phenomena and semantic structures across domains, e.g., temporal knowledge in CALENDAR domain and spatial knowledge in BLOCKS domain. We use the same train/dev/test splits as in the previous work (Wang et al., 2015).

METAQA  METAQA (Zhang et al., 2018) is a large-scale dataset containing more than 400k multi-hop question-answer pairs over MovieQA knowledge base (Tapaswi et al., 2016). Since previous works have achieved ~100% accuracy on its SPARQL annotation (Huang et al., 2021; Shi et al., 2021), we reconstruct METAQA into Cypher as a few-shot learning benchmark. To the best of our knowledge, this is also the first Cypher dataset in the research field of graph query language parsing.

4.2 Metric

We adopt execution accuracy as our metric based on whether the generated logical forms can return correct answers. For queries with multiple answers, we require the execution results to exactly match all the ground truth answers.

4.3 Configurations

For the neural semantic parser, we used the BARTbase model (Lewis et al., 2020) released by Facebook on HuggingFace1. 12 special tokens (e.g., ∼<ES>) were added to the tokenizer vocabulary as the structure indicators for GraphQ IR. We used the AdamW optimizer (Loshchilov and Hutter, 2018) with the learning rate set to 3e−5 and weight decay set to 1e−5 following the default settings.

For the compiler, we used ANTLR (Parr, 2013) version 4.9.2 for analyzing our specified grammar rules and building up the corresponding lexer and parser toolkit. For evaluation, we used Virtuoso2 7.20, SEMPRE3 2.4, Neo4j4 4.4 and KoPL 0.35 as the back-ends respectively for executing the SPARQL, λ-DCS, Cypher and KoPL queries.

Our whole experiments were performed on a

---

1https://huggingface.co/facebook/bart-base
2https://github.com/openlink/virtuoso-opensource
3https://github.com/percyliang/sempre
4https://github.com/neo4j/neo4j
5https://pypi.org/project/KoPL/
Table 2: Experimental results on KQA PRO dataset. Data are categorized into multi-hop inference, qualifier knowledge queries, comparison between several entities, logical union or intersection, count queries for the quantity of entities, verify queries with a boolean answer, and zero-shot queries whose answer is not seen in the training set.

| Baselines                          | Multi-hop | Qualifier | Comparison | Logical | Count | Verify | Zero-shot | Overall |
|-----------------------------------|-----------|-----------|------------|---------|-------|--------|-----------|---------|
| RGCN (Schlichtkrull et al., 2018) | 34.00     | 27.61     | 30.03      | 35.85   | 41.91 | 65.88  | -         | 35.07   |
| BART+SPARQL (Cao et al., 2020b)  | 88.49     | 83.09     | 96.12      | 88.67   | 85.78 | 92.33  | 87.88     | 89.68   |
| CFQ IR (Herzig et al., 2021)     | 87.51     | 81.32     | 95.70      | 90.33   | 86.23 | 92.20  | 87.12     | 88.96   |

| Our Approach                      |           |           |            |         |       |        |           |         |
|-----------------------------------|-----------|-----------|------------|---------|-------|--------|-----------|---------|
| GraphQ IR                         | 90.38     | 84.90     | 97.15      | 92.64   | 89.39 | 94.20  | 91.70     |         |

Table 3: Experimental results on OVERNIGHT dataset. Methods with asterisk (*) means that external data of other domains is utilized to enhance the performance.

| Baselines                          | Bas. | Blo. | Cal. | Hou. | Pub. | Rec. | Res. | Soc. | Overall |
|-----------------------------------|------|------|------|------|------|------|------|------|---------|
| SPO (Wang et al., 2015)           | 46.3 | 41.9 | 74.4 | 54.0 | 59.0 | 70.8 | 75.9 | 48.2 | 58.8    |
| Seq2Action (Chen et al., 2018a)   | 88.2 | 61.4 | 81.5 | 74.1 | 80.7 | 82.9 | 80.7 | 82.1 | 79.0    |
| DUAL (Cao et al., 2019)           | 84.9 | 61.2 | 78.6 | 67.2 | 78.3 | 80.6 | 78.9 | 81.3 | 76.4    |
| 2-stage DUAL* (Cao et al., 2020a) | 87.2 | 65.7 | 80.4 | 75.7 | 80.1 | 86.1 | 82.8 | 82.7 | 80.1    |

| Our Approach                      |       |      |      |      |      |      |      |      |         |
|-----------------------------------|-------|------|------|------|------|------|------|------|---------|
| GraphQ IR                         | 88.2  | 64.7 | 81.6 | 72.0 | 77.6 | 83.3 | 84.9 | 81.6 | 79.5    |
| GraphQ IR*                        | 88.2  | 65.4 | 81.6 | 81.5 | 82.6 | 92.9 | 89.8 | 84.1 | 82.1    |

4.4 Results

As Table 2 illustrates, our proposed approach with GraphQ IR consistently outperforms the previous approaches and has achieved the new state-of-the-art results on KQA PRO dataset. In particular, GraphQ IR exhibits even larger improvements over the baselines under the more complex multi-hop, qualifier and zero-shot task settings. We attribute this to its natural-language-like semantic representations that can be well accepted by the pretrained language model and its formal-language-like syntactic structure that can be flexibly combined or decomposed to achieve better generalization.

As for the benchmark OVERNIGHT, our approaches with GraphQ IR also significantly surpass the baselines, as shown in Table 3. Due to the distinct vocabulary and grammar rules, previous works usually train separate parsers for each task domain (Wang et al., 2015; Chen et al., 2018a). However, as can be observed from the superior performance of GraphQ IR* that get trained on the aggregated data of all eight domains, with an extra layer of IR for unification, domain-specific grammar rules are now consolidated into one universal representation and the training of one domain can thereby benefit from the others’ data.

To further validate our proposed IR’s robustness, we also examine GraphQ IR under compositional generalization and few-shot learning task settings.

Table 4: Experimental results on KQA PRO compositional generalization data split.

|            | Overall | Qualifier | Comparison | Logical |
|------------|---------|-----------|------------|---------|
| BART       | 50.58   | 21.55     | 87.66      | 50.60   |
| CFQ IR     | 50.70   | 25.33     | 93.77      | 50.73   |
| GraphQ IR  | 54.91   | 40.46     | 95.19      | 54.90   |

|            | 1-shot  | 3-shot    | 5-shot    |
|------------|---------|-----------|-----------|
| BART       | 73.93   | 91.99     | 94.37     |
| GraphQ IR  | 72.05   | 93.73     | 95.16     |
| GraphQ IR* | 84.91   | 95.31     | 96.13     |

Table 5: Few-shot learning results on METAQA dataset. GraphQ IR* refers to our model that has formerly trained on KQA PRO dataset. Compositional Generalization Current neural semantic parsers often fail in reaching good compositional generalization, i.e., the capability of generalizing from the known components to produce novel combinations (Pasupat and Liang, 2015; Keysers et al., 2019; Furrer et al., 2020). To measure our IR’s compositional generalization ability, we create a new KQA PRO data split based on the logical form length and test the parsers to generate long
queries based on the short query components seen in the training data. The results are listed in Table 4. Comparing with the plain-BART baseline and the CFQ IR (Herzig et al., 2021) that is specially designed for improving the compositional generalization on SPARQL, GraphQ IR achieves the best performance in overall data as well as in complex task settings, which can be again credited to our IR designs that simplify the redundant semantics and preserve the key structural features.

**Few-shot learning** In practice, it is important for a parser to remain robust in a novel task domain lack of data annotations. Therefore, we reconstruct the METAQA dataset into Cypher, a graph query language commonly used in the industries but rarely studied in previous semantic parsing research works (Seifer et al., 2019), and assess our models under the few-shot learning setting. We adjust the data to ensure only 1, 3 and 5 samples of each question type appear in the training set respectively under the 1-shot, 3-shot and 5-shot settings. The results in Table 5 indicate that our IR can generalize well with very few labelled data. Moreover, the GraphQ IR* model that has in advance trained on KQA PRO data demonstrates the most outstanding performance, especially under the hardest 1-shot setting. This further affirms the necessities of unification. With GraphQ IR, the annotations originally prepared for other graph query languages (e.g., SPARQL) can be now easily transferred to facilitate a novel task domain lack of data.

### 5 Error Analysis

To investigate GraphQ IR’s potentials and bottleneck, we look into the failures of our approach when incorrect logical forms are generated. Out of the total 979 errors in KQA PRO’s test set, we randomly sampled 100 cases and categorized them into 4 types as shown in Table 6:

(1) **Inaccurate data annotation** (28%). The information in user utterance (e.g., “110 minutes”) is inconsistent with the gold SPARQL (e.g., \( v_1 = "110" \)) as annotated in the original dataset. We attribute this type of errors to the dataset rather than the failure of our approach. (2) **Ambiguous query expression** (27%). The semantics of the user utterance may present in more than one way (e.g., “capital_of Uzbekistan”) can convey the same semantics in inconsistent with the gold SPARQL (e.g., \( \text{capital}_1 \text{ capital}_2 \)). The inaccuracy might come from multiple forms of unification. (3) **Unspecified graph structure** (13%). Logical forms of different structures (e.g., “Uzbekistan capital (Tashkent)” and “Tashkent capital (Uzbekistan)”) can convey the same semantics.

### Table 6: The analysis of 4 error types based on the failure cases as occurred in benchmark KQA PRO’s test data.

| Error Type                  | # | # OSC | Example                                                                 |
|-----------------------------|---|-------|-------------------------------------------------------------------------|
| Inaccurate data annotation  | 28| 1     | User utterance: How is the kid’s movie ‘The Spiderwick Chronicles related to John Sayles? |
|                            |   |       | Generated IR: What is the relation from ‘The Spiderwick Chronicles’ to ‘John Sayles’? |
|                            |   | 2     | User utterance: How is the kid’s movie ‘The Spiderwick Chronicles related to John Sayles? |
|                            |   | 3     | User utterance: How is the kid’s movie ‘The Spiderwick Chronicles related to John Sayles? |
|                            |   | 4     | User utterance: How is the kid’s movie ‘The Spiderwick Chronicles related to John Sayles? |

## Table 6: The analysis of 4 error types based on the failure cases as occurred in benchmark KQA PRO’s test data. “# OSC” refers to the number of errors that can be fixed with one step correction on the IR’s structure.
in a directed cycle graph, but some of them contain structures that are absent in a knowledge base. (4) *Nonequivalent semantics* (32%). The output includes incorrect query element *(e.g., string and numerical values)* or structure *(e.g., edges and properties)* that conveys nonequivalent semantics, such as misinterpreting `graduate` to `start_time`.

Overall, 89% of the sampled errors can be simply fixed by the revision of data annotation or one step correction on the GraphQ IR element, demonstrating that our proposed approach with GraphQ IR can generate high quality logical forms and is easy to debug.

6 Related Work

6.1 Semantic Parsing

Semantic parsing aims to translate natural language utterances into executable logical forms, such as CCG *(Zettlemoyer and Collins, 2005)*, λ-DCS *(Liang, 2013; Pasupat and Liang, 2015; Wang et al., 2015)*, SQL *(Zhong et al., 2017; Yu et al., 2020)*, AMR *(Banerescu et al., 2013)*, SPARQL *(Sun et al., 2020)* and KoPL *(Cao et al., 2020b, 2021)*. Most recent works take semantic parsing as a Seq2Seq translation task via encoder-decoder framework, which is challenging due to the semantic and structural gaps between natural utterances and logical forms. To overcome such issues, current semantic parsers usually (1) rely on a large amount of labeled data *(Cao et al., 2020b)*; or (2) leverage external resources for mini the structural mismatch, *(e.g., injecting grammar rules during decoding (Wu et al., 2021; Shin et al., 2021)); or (3) employ synthetic data to diminish the semantic mismatch *(Xu et al., 2020; Wu et al., 2021)*.

Compared with previous works, our proposed GraphQ IR allows semantic parser adapting to different downstream formal query languages without extra efforts and demonstrates promising performance under the compositional generalization and few-shot settings.

6.2 Intermediate Representation

Intermediate representations (IR) are usually generated for the internal use of compilers and represent the code structure of input programs *(Aho et al., 2007)*. Good IR designs with informative and distinctive mid-level features can provide huge benefits for optimization, translation and downstream code generation *(Lattner and Adve, 2004)*, especially in areas like deep learning *(Chen et al., 2018b; Cyphers et al., 2018)* and heterogeneous computing *(Lattner et al., 2020)*.

Recently, IR has also become common in many semantic parsing works that include an auxiliary representation in-between natural language and logical forms. Most of them take a top-down approach and adopt IR similar to natural language *(Su and Yan, 2017; Herzig and Berant, 2019; Shin et al., 2021)*, whereas another category of works instead construct IR based on the key structure of target logical forms in a bottom-up manner *(Wolfson et al., 2020; Marion et al., 2021)*. For example, Herzig et al. designed CFQ IR that rewrites SPARQL by grouping the triples of same elements *(2021)*.

Although these works partially mitigate the mismatch between natural and formal language, they either neglected the structural information requisite for downstream compilation or failed in removing the formal representations that are unnatural to the language models. In contrast, GraphQ IR can benefit from both approaches with its natural-language-like semantic representation and formal-language-like syntactic definition. The unification also saves the huge costs as incurred when preparing separate IR and compiler for each graph query languages in previous works.

7 Conclusion and Future Work

This paper proposes a novel intermediate representation, namely GraphQ IR, for bridging the structural and semantic gap between natural language and graph query languages. Evaluation results show that our approach using GraphQ IR consistently surpasses the baselines on multiple benchmarks covering different formal languages, i.e., SPARQL, KoPL, λ-DCS and Cypher. Moreover, GraphQ IR also demonstrates superior generalization ability and robustness under the compositional generalization and few-shot learning settings.

As an early step towards the unification of semantic parsing, our work opens up a number of future directions. For example, many code optimization techniques *(e.g., common subexpression elimination)* can be incorporated into IR for extra performance boosting. By bringing in multiple levels of IR, our framework may be also extended to support relational database query languages like SQL. Moreover, since current design of GraphQ IR still requires non-trivial manual efforts, the automation of such procedure, *(e.g., in prompt-like manners)*, is definitely worth future exploration.
References

Lakshya A Agrawal, Nikunj Singhal, and Raghava Mutharaju. 2022. A sparql to cypher transpiler: Proposal and initial results. In 5th Joint International Conference on Data Science & Management of Data (9th ACM IKDD CODS and 27th COMAD), pages 312–313.

Alfred V Aho, Monica S Lam, Ravi Sethi, and Jeffrey D Ullman. 2007. Compilers: principles, techniques, & tools. Pearson Education India.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In Proceedings of the 7th linguistic annotation workshop and interoperability with discourse, pages 178–186.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In Proceedings of the 2013 conference on empirical methods in natural language processing, pages 1533–1544.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1415–1425.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1247–1250.

Giovanni Campagna, Silei Xu, Mehrad Moradshahi, Richard Socher, and Monica S Lam. 2019. Genie: A generator of natural language semantic parsers for virtual assistant commands. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 394–410.

Ruisheng Cao, Su Zhu, Chen Liu, Jieyu Li, and Kai Yu. 2019. Semantic parsing with dual learning. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 51–64.

Ruisheng Cao, Su Zhu, Chenyu Yang, Chen Liu, Rao Ma, Yanbin Zhao, Lu Chen, and Kai Yu. 2020a. Unsupervised dual paraphrasing for two-stage semantic parsing. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 6806–6817.

Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. 2020b. Kqa pro: A dataset with explicit compositional programs for complex question answering over knowledge base. arXiv preprint arXiv:2007.03875.

Shulin Cao, Jiaxin Shi, Zijun Yao, Xin Ly, Jifan Yu, Lei Hou, Juan-Zi Li, Zhiyuan Liu, and Jinghui Xiao. 2021. Program transfer for answering complex questions over knowledge bases.

Bo Chen, Le Sun, and Xiamei Han. 2018a. Sequence-to-action: End-to-end semantic graph generation for semantic parsing. arXiv preprint arXiv:1809.00773.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018b. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pages 578–594.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world’s response. In Proceedings of the fourteenth conference on computational natural language learning, pages 18–27.

Scott Cyphers, Arjun K Bansal, Anahita Bhiwandiwalla, Jayaram Bobba, Matthew Brookhart, Avijit Chakraborty, Will Constable, Christian Convey, Leona Cook, Omar Kanawi, et al. 2018. Integraph: An intermediate representation, compiler, and executor for deep learning. arXiv preprint arXiv:1801.08058.

Marco Damonte and Emilio Monti. 2021. One semantic parser to parse them all: Sequence to sequence multitask learning on semantic parsing datasets. arXiv preprint arXiv:2106.04476.

Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdellkawi, and Jens Lehmann. 2019. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In International semantic web conference, pages 69–78. Springer.

Michael H Fischer, Giovanni Campagna, Euirim Choi, and Monica S Lam. 2021. Diy assistant: a multimodal end-user programmable virtual assistant. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, pages 312–327.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärtl. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. arXiv preprint arXiv:2007.08970.

Jonathan Herzig and Jonathan Berant. 2019. Don’t paraphrase, detect! rapid and effective data collection for semantic parsing. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 3810–3820.

Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. 2021. Unlocking compositional generalization in pre-trained models using intermediate representations. arXiv preprint arXiv:2104.07478.
Wonseok Hwang, Jiyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikiSQL with table-aware word contextualization. arXiv preprint arXiv:1902.01069.

Aishwarya Kamath and Rajarshi Das. 2018. A survey on semantic parsing. In Automated Knowledge Base Construction (AKBC).

Daniel Keysers, Nathanael Schärl, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2019. Measuring compositional generalization: A comprehensive method on realistic data. arXiv preprint arXiv:1912.09713.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic ccg grammars from logical form with higher-order unification. In Proceedings of the 2010 conference on empirical methods in natural language processing, pages 1223–1233.

Chris Lattner and Vikram Adve. 2004. Llvm: A compilation framework for lifelong program analysis & transformation. In International Symposium on Code Generation and Optimization, 2004. CGO 2004., pages 75–86. IEEE.

Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Plenarier, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2020. Mlir: A compiler infrastructure for the end of moore’s law. arXiv preprint arXiv:2002.11054.

Mike Lewis, Yinhua Liu, Naman Goyal, Marjan Ghazvininejad, Abdulrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7871–7880.

Zhuang Li, Lizhen Qu, and Reza Hafifari. 2020. Context-dependent semantic parsing: a survey. In International Conference on Computational Linguistics 2020, pages 2509–2521. Association for Computational Linguistics (ACL).

Percy Liang. 2013. Lambda dependency-based compositional semantics. arXiv preprint arXiv:1309.4408.

Percy Liang, Michael I Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. Computational Linguistics, 39(2):389–446.
Table 7: A more complete set of GraphQL IR grammar rules that covers the common graph query patterns. Italic words refer to the terminal symbols. For simplicity, here we omit the special production rules for handling the corner cases.
User utterance:
friends of people who joined their jobs before 2005

GraphQ IR sequence:
what is <ES> <ES> <C> person </C> </ES> that <R> friend </R> backward to
<ES> <C> employee </C> <ES> ones
whose <A> employment start date </A> at most year <V> 2004 </V> </ES> </ES> </ES>

GraphQ IR abstract syntax tree:

Lambda DCS abstract syntax tree:

Lambda DCS sequence:
( call @listValue ( call @filter ( call @getProperty ( call @singleton en.person )
( string ! type ) ) ( call @reverse ( string friend ) )
( string = )
( call @getProperty ( ( lambda s ( call @filter ( var s )
( call @ensureNumericProperty
( string employment_start_date )
( string <= )
( call @ensureNumericEntity ( date 2004
-1 -1 ) )
( call @domain ( string employee ) )
( string employee ) ) ) )

Figure 3: A user query in OVERNIGHT. As aforementioned in Section 3, the neural semantic parser first translates the input utterance into the GraphQ IR. The front-end of compiler then parses the GraphQ IR sequence into an abstract syntax tree, which is subsequently transformed into the corresponding λ-DCS sequence by the compiler back-end.
User utterance:
Which has less elevation above sea level, Rome that is the filming location of To Rome with Love or Lisbon which is the twinned administrative body of Santo Domingo?

GraphQ IR sequence:
which one has the smallest <A> elevation above sea level </A> among <ES> <ES> <E> Rome </E> (the ones that <R> filming location </R> forward to <ES> To Rome with Love </ES> or <ES> <E> Lisbon </E> (the ones that <R> twinned administrative body </R> forward to <ES> Santo Domingo </ES>)) <ES> </ES>)

GraphQ IR abstract syntax tree:

SPARQL abstract syntax tree:

SPARQL sequence:
SELECT ?e WHERE { { ?e <pred:name> "Rome" . ?e_1 <filming_location> ?e . ?e_1 <pred:name> "To Rome with Love" } UNION {?e <pred:name> "Lisbon" . ?e_1 <twinned_administrative_body> ?e . ?e_1 <pred:name> "Santo Domingo" } ?e <elevation_above_sea_level> ?pv . ?pv <pred:value> ?v . } ORDER BY ?v LIMIT 1

Figure 4: A user query in KQA PRO. Similarly, the compiler parses the generated GraphQ IR sequence into an abstract syntax tree, then transform its tree structure into the corresponding SPARQL sequence.