Seq2Cypher: A Dataset Mapping Approach from Relational Database to Property Graph Database for Natural Language Query

Ziyu Zhao[1,2], Wei Liu[1,2], Tim French[1,2], and Michael Stewart[1,2]

[1] UWA NLP-TLP Group https://nlp-tlp.org/
[2] School of Physics, Mathematics and Computing,
The University of Western Australia, Australia
ziyu.zhao@research.uwa.edu.au
{wei.liu, tim.french, michael.stewart}@uwa.edu.au

Abstract. Text-to-Cypher is the task that translates natural language queries into Cypher queries, given a property graph database. It is known as semantic parsing task, where a model predicts (neural network based) or generates (rules based) a machine-readable formal query, which can be executed against a database that stores structured data. Most of related work mainly focus on text-to-SQL in relational database setting, and there is little available dataset for semantic parsing method in the context of graph database. Addressing the lack of dataset for natural language queries and the corresponding Cypher queries, is a key contribution for Text-to-Cypher queries translation. To bridge this gap, we propose an approach called Seq2Cypher, mapping from relational databases to property graph database, as well as SQL queries to Cypher queries, for natural language queries. We evaluate our proposed Seq2Cypher on Spider benchmark dataset, with execution accuracy metrics to measure the performance. Furthermore, we train and evaluate a baseline model on our proposed dataset. The experimental results show that...

Keywords: Neural Semantic Parsing · Text-to-Cypher

1 Introduction

There is an increasing amount of data stored in graph databases because its flexibility in schema and better performance in capturing relations between entities. This is especially true for knowledge graphs constructed from information extraction over textual data in technical domains, e.g. ECHIDNA [**?**]. However, navigating large technical knowledge graphs can be quite challenging for users without coding experience. Therefore, natural language queries become popular for simplified human knowledge graph interaction, and at times even essential due to hands-free requirement for workplace safety. The research in semantic parsing is mostly focusing on automated translation between natural languages and relational database access using SQL (e.g., Spider and WikiSQL) [**?**]. There is a lack of datasets for learning models to map natural language to graph database query language.

2 Related Work

Our work lies at the intersection of neural semantic parsing and graph queries. We review related work in neural semantic parsing models for Text-to-SQL and current progress in translating text to various graph query languages.

Neural Semantic Parsing Neural semantic parsing in an end-to-end manner with pre-trained language models (PLMs), uses specialized encoders [**?**] by taking structured data and text data as input, and customized decoders, such as constrained decoder [**?**] and ranking [**?**], to generate formal query languages [**?**], owing to the different types of structured knowledge, such as relational database and knowledge graph. Unlike task-specific architecture on semantic parsing task, *large-scale multi-task learning with PLMs*, on the other hand, is proposed to pefix-tune a unified framework in the benefit of knowledge or parameters sharing [**?**].

In the research line for Text-to-SQL task setting, there are many available datasets[3], such as Spider [15] for semantic parsing, WikiSQL [16] for question answering, and CoSQL [14] for querying dialogue systems. There are no overlap between questions or databases on training, development and test splits of Spider and CoSQL. Among them, Spider is the first dataset covering cross domain and a significant amount of complex SQL queries. On top of Spider, Yu et al. [15] defines a new semantic parsing task and proposed SyntaxSQLNet model. The model is fed questions and database schema items, and generates a syntax tree rather than a SQL query. In Spider, Schema linking mechanism

One of the research questions is how to effectively incorporate DB content in the context of text-to-SQL task setting. Lin et al. [6] make use of `anchor text`, which refers to the mentioned DB content in questions, to link to corresponding relational DB fields, and the value of targeted DB fields are saved to value sets called `picklists`. For each question, during encoding process, they performs fuzzy string match between the question and the picklist of each field in the DB, and concatenate matched DB fields values with the serialization of the DB schema. Take the input format of BERT as an example, the typical hybrid question-schema serialization is show below, where

$$X = [CLS], Q, [SEP], [T], t_1, [C], c_{11}..., c_{1|T_1|}, [T], t_2, \\ [C], c_{21}, [V], v_{21}..., v_{2|T_M|}, c_{31}..., c_{N|T_N|}, [SEP] \quad (1)$$

Researchers have show the model performance without taking into account the value in tables dramatically drop

Another question is *constrained decoding*.

RCSQL [**?**] PICARD [9]

Intermediate representation(IR) for SQL address the mismatch issue between utterances and formal query languages. Natural SQL [1] SemQL [**?**] RAT-SQL [**?**]

Neural semantic parsing models for relational databases are widely applied in the natural language interface applications, for example question answering [**?**], dialogue system paradigm [**?**].

[3] Available in the UnifiedSKG [13] site: https://unifiedskg.com/benchmarks/

Machine Translation from Text to Graph Queries Graph databases with a continuously updated external knowledge source are useful for representing, organising and analysing data. It is efficient for users to aggregate over not directly interconnected information in knowledge graphs (KGs) via a simple graph query. Although existing related work mainly focuses on Text-to-SQL tasks, there are a few research working on machine translation from text to graph query language, such as GraphQL[4] and SPARQL query[5]. Most of neural end-to-end approaches rely on an annotated training set training a model to translate natural language queries into executable database queries, Irina and Anton [8] reduce the dataset burden by training a neural semantic parser from text to an intermediate representations, and a non-trainable transpiler to a structurally closer formal database query language SPARQL. Ni et al. [7] trained a Text2GraphQL task model generating GraphQL statements for graph database retrieval in medical consultation chatbot application. Their end-to-end pipeline model is composed of a XLM [4] by inserting a pre-trained Adapter [3] plug-in as the encoder, and a Pointer Network [11] as the decoder. They evaluated the model on an annotated medical QA knowledge graph dataset curating from a large online medical QA forum, as well as the counterpart datasets (Spider, ATIS, GeoQuery, and 39.net)[6]. Typically, a GraphQL Abstract Syntax Tree (AST) schema is generated via a PostgreSQL[7] database which is mapped from a Spider SQLite dump. The ATS is then encoded as a GraphQL query in the string format, but some SQL queries can not be transferred to GraphQL queries within Hasura[8], such as `GROUP BY` clauses.

3 Task Description

We define a Text-to-Cypher task, translating questions in natural language to Cypher queries that can execute against a given a property graph database hosted in Neo4j[9]. There is few prior work in this area, and it is difficult to collect graph databases with complex schema. We propose a dataset to work with our model which is mapped from Spider, a benchmark for Text-to-SQL over cross domain relational databases.

Assumptions In order to focus on the more fundamental part of the semantic parsing model from text to Cypher query language, we make the following assumptions. Firstly, in this paper, as same as the one made in Spider [15], we do not evaluate the model performance on generating field values, instead we slot filling the targeted values out of a given list of gold values for each

[4] `GraphQL`: https://graphql.org/learn/
[5] `SPARQL`: https://www.w3.org/TR/rdf-sparql-query/
[6] Their proposed semantic parsing method and datasets for Text-to-GraphQL is not publicly available.
[7] `PostgreSQL is hosted in Hasura`: https://www.postgresql.org/
[8] `Hasura is a GraphQL API`: https://hasura.io/
[9] `Neo4j`: https://neo4j.com/docs/getting-started/current/get-started-with-neo4j/

question. Secondly, `intersect` and `except` queries are excluded, which can be simplified into nested queries somehow. Thirdly, we assume most of the relational database constraints are self-contained in the mapped graph database, but all table and column names in SQL queries are checked and aligned with the corresponding elements in the graph database, such as node labels, relationship types and properties, when being translated into the Cypher queries. For example, "department_id" is automatically converted to "Department_ID" according to the graph meta data, which is constructed directly from relational database SQLite schema, due to the fact that Cypher queries are case-sensitive. The last but not the least, we explicitly encode primary key and foreign key (PK-FK) constraints by introducing the schema-induced graph edges.

Problem Definition Given a set of natural language questions $\mathcal{Q}$, and a relational database schema $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, \mathcal{V} \rangle$, where T is a set of all table names, C and V denote each table's corresponding set of column names and column's values. Our first goal is to map the relational database dump $\mathcal{S}$ to a graph database $\mathcal{G} = \langle \nu, \varepsilon \rangle$, and then to generate a corresponding Cypher query. Graph node labels and graph relationship types are described by table names. A set of $\mathcal{C}$ is called `properties` in the context of graph database. The graph nodes set $\nu = \langle \mathcal{C} \cup \mathcal{V} \rangle$ are obtained from the columns and values of columns. Each table's name is with respect to the label of a set of graph nodes ν. The edge ε are defined by the pre-existing PK-FK constraints of $\mathcal{S}$. Considering a pair of $\langle q, g \rangle$, where $q \in \mathcal{Q}$ and $g \in \mathcal{G}$, we then aim to translate q to a Cypher query.

4 From Relational Database to Property Graph

4.1 Raw Dataset

Several public datasets are available for natural language utterance to SQL in querying relational databases. The goal of text-to-SQL models is to generate the correct SQL query for a question in natural language and table schema (i.e., table column names), without/with using the records of tables (i.e., table content). Two of the most widely used are WikiSQL [16] and Spider [15]. Compared with `WikiSQL`, `Spider` is much richer in the variants of both natural language queries and SQL queries, in terms of complex queries. For example, Spider contains about twice more nested queries and 10 times more `GROUP BY (HAVING)` and `ORDER BY (LIMIT)` statements, especially `JOIN ON` statements which are mapped to graph edges. While all SQL queries in `WikiSQL` are simple, and each database is only a simple table without any foreign key. The statistics of Spider is listed in Table 1, including the average, minimum and maximum length of three inputs, namely natural language query (NLQ), structured tables' headers (STH) and SQL queries. We carry out two stages to pre-process Spider benchmark (Section 4.2 and 4.3) and a evaluation process (Section 4.4), to work with our model.

Table 1: Data statistics of Spider, including three inputs in Spider. `NLQ` and `STH` stand for natural language query and structured tables' heads.

				NLQ			**STH**			**SQL**		
	# Q	**# SQL**	**# DB**	avg	min	max	avg	min	max	avg	min	max
Train	8,659	4,730	140	13.9	1.0	45.0	124.7	31.0	512.0	20.4	3.0	78.0
Dev	1,034	564	20	14.2	5.0	34.0	103.4	38.0	257.0	18.1	6.0	54.0

4.2 Graph Population from Relational Database Schema

Mapping Strategy We propose to construct a property graph database directly from any relational database schema, where each relational database contains several tables. Each table is mapped either into graph nodes or graph edges, and the rows of a table are the properties of the target graph component. Relational primary key (PK) constraints and foreign key (FK) constraints are exploited to facilitate the automatic mapping decision, and some inconsistent PK and FK constraints are identified and amended during the mapping exercise. However, we do not check the the rest three constraints (domain constraint, tuple uniqueness constraint and entity integrity constraints), and assume they are consistent. In addition, due to the limitation of Neo4j, we only represent `one-to-one` and `one-to-many` (or `many-to-one`) relationships in the aimed graph, while `many-to-many` type of relations are resolved to the other two equivalent relations. Figure 1 shows the strategy of mapping process.

Taking `department_management.db` as an example, it contains three tables, i.e. table `department` with one PK *Department_ID*, table `head` with a PK *head_ID*, and table `management` with a compound PK (*Department_ID*, *head_ID*) in `management`. The one-to-one relationship lies on 's PK-FK constraints, where *Department_ID* is also a FK referred to table `department`, and the FK *head_ID* is referenced to `head`. Based on our mapping schema, both table `department` and `head` become graph nodes, and table `management` turns into graph edges. Specifically, the rows of each table are graph properties along with each table name as their graph components' label accordingly. Figure 2 visualises the connected graph components among three tables.

Graph Statistics and Analysis Table 2 presents the statistics of projected graph. Considering the computation capacity, we drop any relational database, if the number of rows of any table exceeds a specific threshold. We set the threshold as 4,000, and it ends up keeping 129 out of 166 relational databases. The number of tables are denoted as schema data, and instances stand for graph components.

Based on the leaderboard[10], there are around 104 published papers using Spider dataset. It is important to understand the quality of the dataset itself. Therefore, besides the mapping exercise according to our proposed mapping strategy, we also carry out and amend data error from the perspective of consistency.

[10] `Spider benchmark leaderboard`: https://yale-lily.github.io/spider

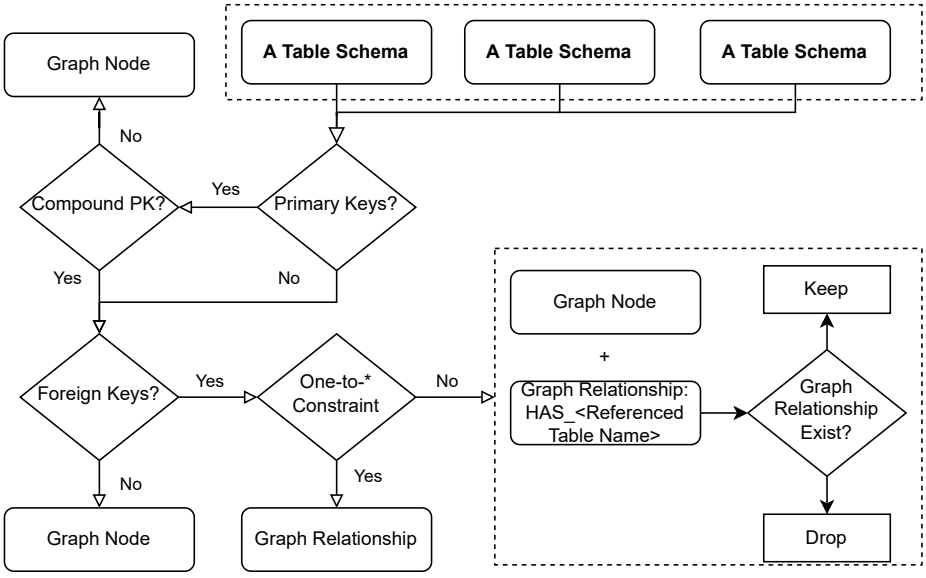

Fig. 1: The mapping strategy from a relational database dump, to a property graph. PK is short for `primary key`. `HAS_<Referenced Table Name>` is the format of a curated graph relationship's type, where `Referenced Table Name` refers to a graph destination node with the label of the referenced table's name. `One-to-*` denotes one-to-one and one-to-many (or a vice versa) relationship in relational database. Note, we construct a mapped graph in the sequence of graph nodes, graph relationships, and curated graph relationships. We ignore a table without any primary key nor a foreign key.

1) Graph Node Label Differentiation Issue We observe one of the disadvantages using Neo4j graph database, is that it can not partition different domains automatically. For example, there are two tables both named `singer` in two different relational database dumps, `singer` and `concert_singer`. If we host both the two domains data, it brings execution error into the translation process from a SQL query to a Cypher query. The first example in Table 3 illustrates that, even though the converted Cypher query is correct, the execution answer is not exactly same with the gold SQL execution answer. If we zoom out, the different one `'Justin Brown'` belongs to `concert_singer` domain rather than `singer` domain. The error would disappear if we only host `singer` in Neo4j. Certainly, the error can be eliminated via renaming each table's name in the `domain_name.table_name` format, such as the graph edge `department_management.management` in Figure 2.

2) No Primary Key and Duplicate Rows Problem; 3) No table Content Issue Apart from the graph node label differentiation issue, we also identify and amend other relational database problems existing in Spider benchmark. One of the most significant one is *no primary key and duplicate rows problem*. There are

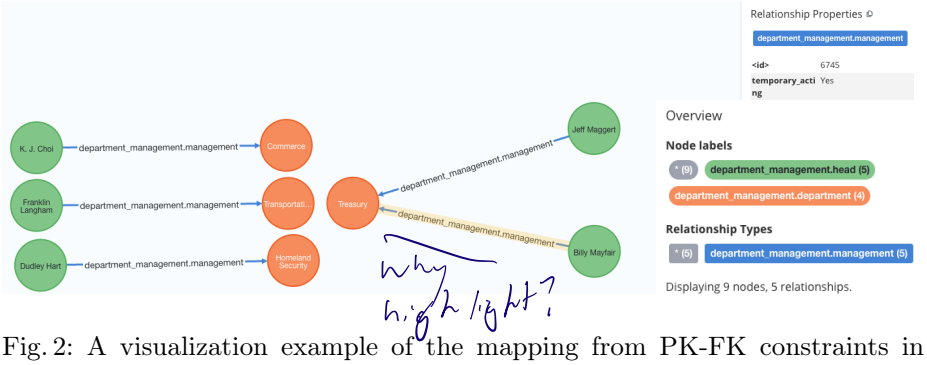

Fig. 2: A visualization example of the mapping from PK-FK constraints in `department_management` relational dump. The graph nodes in grey denote the information in table `head`, and the nodes in orange are mapped from table `department`. They all show the names of each head and department. The graph edges in green result from table `management`.

Table 2: Statistics of the graph constructed from Spider benchmark.check the terminologies.

		# Triples	
Graph Category	**# Table-to-Node**	**# Table-to-Edge**	**# HAS_***
Schema Data	606	95	215
Instances	11,797	3,204	12,044

95 tables out of 876 (10.8%) that do no have any primary key, among which 11 tables out of 876 (1.3%) exist duplicate rows. For this particular issue, if any PK-FK constraints obeys our uniqueness quantification, where the one and only the number of PK-FK constraints satisfy the condition that equals 2, we apply the table-to-edge strategy likewise the case of compound primary key, and the duplicate rows are automatically removed during the graph construction process. The other one is *no table content issue*. There are 7 relational databases, any table of which only contain table columns but no table content. We set the value of each table column empty, based on the restriction of creating graph nodes in Neo4j.

4) Data Consistency Maintenance Additionally, we check and maintain the consistency of primary key and foreign key as well. For example, in `musical` domain schema, table `actor` is referenced to itself. After checking the PK-FK constraint, i.e. `FOREIGN KEY (“Musical_ID”) REFERENCES “actor”(“Actor_ID”)`, it is more straightforward to reference to table `musical`, insteaf of a self-referencing relationship. Above all, we remove foreign keys from the property set of an edge for the sake of not violating data consistency when changing the related primary keys of nodes. In Figure 2, we only keep the `temporary_acting` as the property of edge `department_management.management` (see top right), and abandon

Table 3: The examples for error analysis. `EA` is short for execution accuracy.

	A nested query example.
NLQ:	`What is the sname of every sing that does not have any song?`
SQL query:	`SELECT Name FROM singer WHERE Singer_ID NOT IN (SELECT Singer_ID FROM song)`
SQL query EA:	**[['Alice Walton'], ['Abigail Johnson']]**
Cypher query:	`MATCH (si:singer) WHERE NOT (si:singer)-[]-(:song) RETURN si.Name`
Cypher query EA:	**[['Justin Brown'], ['Alice Walton'], ['Abigail Johnson']]**
	An example that duplicate elements are dropped automatically.
NLQ:	`Which skill is used in fixing the most number of faults? List the skill id and description.?`
SQL query:	`SELECT T1.skill_id, T1.skill_description FROM Skills AS T1 JOIN Skills_Required_To_Fix AS T2 ON T1.skill_id = T2.skill_id GROUP BY T1.skill_id ORDER BY count(*) DESC LIMIT 1`
SQL query EA:	**[[3, 'TV, Video']]**
Cypher query:	`MATCH (T1:'assets_maintenance.Skills')-[T2:'assets_maintenance.Skills_Required_To_Fix']-() WITH T1.skill_description AS skill_description, count(T1.skill_id) AS count, T1.skill_id AS skill_id RETURN skill_id,skill_description ORDER BY count DESC, LIMIT 1`
Cypher query EA:	**[[2, 'Mechanical']]**

`department_id` and `head_id`. At last, We use the "Repair" scenario of the mapping process (see Figure 3). In this scenario, the expected data statistics is filtered out of the whole data set based on a threshold. The mapping result is loaded and validated. Particularly, we use `APOC Procedures`, such as `apoc.meta.graph()`, to check the statistics of the obtained graph with our expected relational database data in terms of labels of nodes, types of edges and properties. Next, a difference of the two is identified and revalidated, and the mapping process is rechecked and the graph is rebuilt until there is no difference between the expected graph database statistics and the actual one. This aims to simulate the workload of a database transformation process.

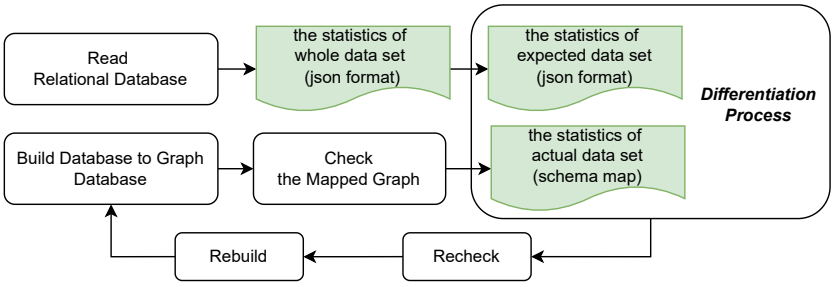

Fig. 3: Phases of the Check Mapping Consistency and Repair Scenario.

4.3 SQL to Cypher

Translation Guidance Aim at avoiding mismatch issues between Cypher queries and natural language queries from dataset perspective, we carry out the translation process from SQL queries to Cypher queries aligned with the guidance based on the following questions.

Q1: Can we simplify SQL queries with Cypher queries on the constructed property graph? SQL is designed for effectively querying relational databases, rather than representing the counterfactual parts of input in natural language [1], it is challenging for neural network models to generate complex SQL queries keywords (for example `JOIN ON`, `GROUP BY` and `HAVING`, together with the corresponding schema items w.r.t. relational databases. Take the question in *Example1* shown below as an instance, there are `HAVING` and `GROUP BY` statements in the given gold SQL query. Although some existing Text-to-SQL models proposed various intermediate representation (IR) [1,?], either working on schema linking mechanism, or removing the aforementioned unnatural SQL keywords, the problems are not eliminated, and it requires post-processing before the final SQL queries are executable. However, this issue is addressed during the process of constructing an equivalent property graph from relational database, where all the tables' foreign keys constraints utilised in `JOIN ON` statements, are projected to graph relationships. Therefore, from the perspective of simplifying SQL target query language representation, we argue the proposed dataset are potentially more efficient, compared with gold SQL queries and other IR representation.

Example1. Consider a question, "Which department has more than 1 head at a time? List the id, name and the number of heads", and the goal SQL query is in List 4.1. The translated Cypher query is shown on List. 4.2, and its result on Fig. 4. Firstly, it reduces the `JOIN ON` statement to a simple graph path pattern in `MATCH` statement. Secondly, it retrieves the language of messages that were written in the `GROUP BY` and `HAVING` statements. If that message was a condition, the outermost of the original message is also retrieved. All the mentioned schema fields are listed in `WITH` statement, and the content of mentioned conditioned schema fields are added in `WHERE` statement. Since all PK-FK constraints are transferred to graph relationship, `JOIN ON` statements are then reduced to graph paths specification. It relieves the mismatch between natural language queries and the corresponding Cypher queries, which is a key challenges in the task text-to-SQL translation.

```
SELECT T1.department_id, T1.name, count(*)
FROM management AS T2 JOIN department AS T1
ON T1.department_id=T2.department_id
GROUP BY T1.department_id
HAVING count(*) > 1
```

List. 4.1: The gold SQL query in *Example1.*

```
MATCH ()-[T2:management]-(T1:department)
WITH T1.Department_ID AS Department_ID,
count(*) AS count, T1.Name AS Name
WHERE count  > 1
RETURN Department_ID, Name, count
```

List. 4.2: Cypher query for *Example1.*

Fig. 4: Execution Result.

Return Field	Value
Department_ID	2
Name	'Treasury'
count	2

Q2: Is Cypher query easier to infer from natural language queries than SQL query?

Example2. Given a question, "How many departments are led by heads who are not mentioned?", and the goal SQL query is in List 4.3. nested nested query

```
SELECT count(*) FROM department
WHERE department_id NOT IN
(SELECT department_id FROM management)
```

List. 4.3: The gold SQL query in *Example2*.

```
MATCH (d:department)
WHERE NOT (d:department)-[:management]-()
RETURN count(*)
```

List. 4.4: Cypher query for question in *Example2*.

Fig. 5: Execution Result.

Return Field	Value
count	11

Q3: How to achieve the equivalent representation of SQL clauses with Cypher clauses?

Implementation Details The pseudo-code of SQL-to-Cypher process is described in algorithm 1. At this stage, we filter out *intersect* or *except* queries. They can be simplified into nested queries and we leave these to future work. Our web-based interface[11] implementation is available at `github`. It supports users to select a domain, and write any valid SQL query, our interface is able to translate it into a corresponding Cypher query, and return the execution answers of both two queries.

4.4 Mapping Evaluation and Analysis

We evaluate our proposed approach using execution accuracy metric. Execution accuracy (EA) measures if the execution result of the obtained Cypher query against the obtained property graph database is the same as the gold SQL queries executed on relational databases, accordingly. Since the quality of executing a translated Cypher query also depends on the accuracy of the constructed property graph database from the relational database dumps, EA metric reflect the aforementioned mapping process implicitly. In Table 4, we present the overall performance and several relational databases as well. The score on development set is lower than training set, owing to the different data store discipline. Duplicate table rows in relational databases are automatically dropped during the graph components, and it results in the different execution results. For instance, in Table 3, the second example illustrate this kind of cases. In `assets_maintenance` relational schema, there are two consecutive duplicates

[11] The proposed web-based interface is adapted from Li et al. [5]

Algorithm 1: Generate Cypher queries from SQL queries, where SQLClauses* is a SQL clauses list in execution order, `[from, where, select, orderby, limit, offset?]`, and CypherClauses** is the corresponding list `[match, where, return, orderby, limit, offset?]`.

Input: E is one SQL query. "·" refers to graph schema.
Output: G is the obtained Cypher query with respect to E.
Function *SQL2Cypher*$(F(E,\cdot))$
  $G \leftarrow \{\}$ `// initial set of generated Cypher clauses`
  $S \leftarrow \{SQLClauses^{*}\}$ `// queue for SQL clauses keywords`
  $Q \leftarrow \{CypherClauses^{**}\}$ `// queue for Cypher clauses keywords`
  foreach $s \leftarrow$ DEQUEUE(S) **do**
    $Z \leftarrow$ DEQUEUE(Q)
    if *e has nested_ query* **then**
      $Z \leftarrow \{z \in (F(nested\_query, s)) \mid F(e,\cdot)\}$
        `// all outer most level statements regarding CLAUSE Q`
    else if *e contains aggregate function call* **then**
      $Z \leftarrow$ WITH_AS$(Z, level(s+1), F(e,s))$
        `// manipulate the output before it is passed on to the following query parts`
        `// aggregation allowed only at the outermost level`
    else
      $Z \leftarrow F(e,s))$ `// obtain Cypher query statement w.r.t. s`
    ENQUEUE(G, Z)
  $G \leftarrow G \cup \{Z\}$ `// append to the Cypher clause keyword`
  $S = \varnothing$ `// SQLClauses is empty`
  return G

in table `Skills_Required_To_Fix`, so they are merged during the construction of graph components.

5 Model

Even though T5-3B or T5-Large based models have been proved outperforming, it is computationally expensive in fine-tuning and inference. Inspired by the experiments setting [13], we only apply a T5-based transformer, and follow its dataset loading format.

5.1 Baseline and Evaluation Metrics

Spider is a zero-shot setting, and Shaw et al. [10] showed that a pre-trained T5 family models can not only learn the text-to-SQL tasks, but also generalize to unseen domain. In addition, due to the limitation of our computing capacity, we are encouraged to setup two baselines in the task of Text-to-SQL: T5-base

Domain	Count	Execution Accuracy (EA)
musical	38	1.0
department_management	15	1.0
hospital_1	90	0.84
asserts_maintenance	28	0.82
Train	7,398	0.6
Dev	795	0.77

Table 4: We report the execution accuracy (EA) of our ablation domain, as well as the whole train and development (Dev) splits.

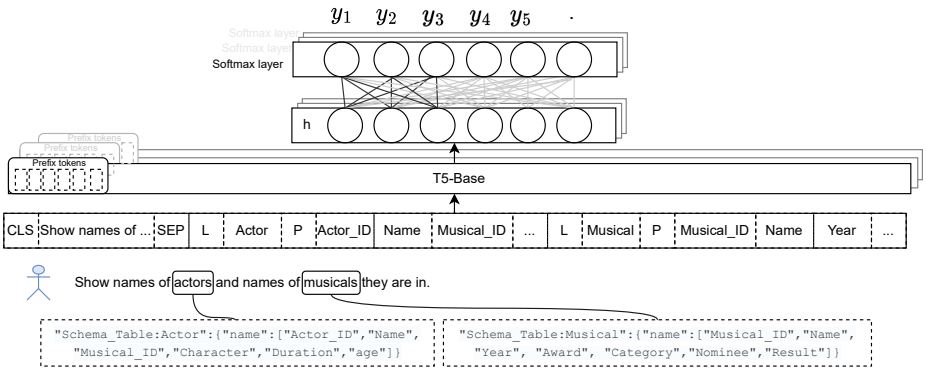

Fig. 6: An illusion of T5-base model with prefix-tuning setting.

fine-tuning and prefix-tuning models used in UnifiedSKG framework [13]. At the time of writing, we do not have Spider test set, so only the results on train and development sets are reported. We also report the performance of `Text-to-GraphQL`[12]. `Exact set match` and `execution accuracy` are used to measure the models performance. `Exact set match` compares the predicted and the ground-truth SQL queries by parsing both into a normalized data structure [9].

5.2 Experiment Analysis

Table 5

6 Conclusion

In this paper, we proposed a mapping approach to use the cross-domain benchmark Spider as raw dataset for both generating executable Cypher queries and a

[12] It is the most close work. It would be interesting to test our dataset on their proposed model, or run our baseline models on their counterpart Spider1.0 for GraphQL. We have emailed them for the access to their codes and dataset, but do not receive any respond yet at the time of writing this paper.

Table 5: Exact set match (top), execution accuracy (middle) on the Spider development (dev) and test sets. BERT_{L} denotes $\text{BERT}_{\text{LARGE}}$. ♠ denotes approaches using DB content. ♡ denote approaches generating executable output. At bottom, since we do not have the Spider test set when writing this paper, we report the average execution accuracy on the Spider train, and execution accuracy (exact match) on dev sets respectively.

Model	**Dev**	**Test**
Text-to-GraphQL + Adaptor [7]	*75.8*	*72.3*
BRIDGE + BERT_{L} (ensemble) [6]	71.1	67.5
RAT-SQL v3 + BERT_{L} ♠ [12]	69.7	65.6
IRNet + BERT [2]	61.9	54.7

Model	**Dev**	**Test**
Text-to-GraphQL + Adaptor [7] ♠ ♡	-	*76.2*
BRIDGE + BERT_{L} (ensemble) [6] ♠ ♡	70.3	68.3

Model	**Train/avg EA**	**Dev/EA (EM)**
Text-to-SQL + T5_{base} (fine tune)♠ ♡	61.36	62.67 (60.06)
Text-to-SQL + T5_{base} (prefix tune)♠ ♡	63.39	64.41 (62.38)
Ours + T5_{base} (fine tune) ♠ ♡		
Ours + T5_{base} (prefix tune) ♠ ♡		

graph database, given natural language queries. Using Seq2Cypher is beneficial because any relational database schema can be directly mapped to a graph database hosted in Neo4j, and recursively parse any SQL query into an equivalent Cypher query, which have shown being relatively naturally representations to questions in natural language. We also set a simple T5-base baseline model for future work. What's worth mentioned, we observe that there are around 5% tables that can be converted into hyperedges. Each hyperedge may consist of more than two nodes, for example, in `college_3` schema, table Enrolled_in is constrained to other three different tables respectively, hence each row in the aforementioned table acts as a hyperedge, where its column `StuID` is referenced to reference table `Student`, `Grade` is used to link to table `Gradeconversion`, and `CID` is constrained to table `Course`. We reckon our proposed dataset potentially can be used for effective hyperedge representation learning.

References

1. Gan, Y., Chen, X., Xie, J., Purver, M., Woodward, J.R., Drake, J., Zhang, Q.: Natural SQL: Making SQL easier to infer from natural language specifications. In: Findings of the Association for Computational Linguistics: EMNLP 2021. pp. 2030–2042. Association for Computational Linguistics, Punta Cana, Dominican Republic (Nov 2021). https://doi.org/10.18653/v1/2021.findings-emnlp.174, `https://aclanthology.org/2021.findings-emnlp.174`
2. Guo, J., Zhan, Z., Gao, Y., Xiao, Y., Lou, J.G., Liu, T., Zhang, D.: Towards complex text-to-sql in cross-domain database with intermediate representation. arXiv preprint arXiv:1905.08205 (2019)

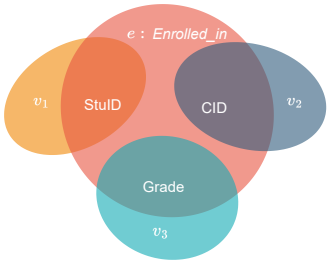

Fig. 7: An illustion of `Enrolled_in` hyperedge denoted as e. v_1, v_2, v_3 represent graph nodes labelled as `Student`, `Course` and `Gradeconversion` accordingly.

3. Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., Gelly, S.: Parameter-efficient transfer learning for nlp. In: International Conference on Machine Learning. pp. 2790–2799. PMLR (2019)
4. Lample, G., Conneau, A.: Cross-lingual language model pretraining. arXiv preprint arXiv:1901.07291 (2019)
5. Li, S., Yang, Z., Zhang, X., Zhang, W., Lin, X.: Sql2cypher: Automated data and query migration from rdbms to gdbms. In: International Conference on Web Information Systems Engineering. pp. 510–517. Springer (2021)
6. Lin, X.V., Socher, R., Xiong, C.: Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, November 16-20, 2020 (2020)
7. Ni, P., Okhrati, R., Guan, S., Chang, V.: Knowledge graph and deep learning-based text-to-graphql model for intelligent medical consultation chatbot. Information Systems Frontiers pp. 1–20 (2022)
8. Saparina, I., Osokin, A.: SPARQLing database queries from intermediate question decompositions. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. pp. 8984–8998. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic (Nov 2021). https://doi.org/10.18653/v1/2021.emnlp-main.708, `https://aclanthology.org/2021.emnlp-main.708`
9. Scholak, T., Schucher, N., Bahdanau, D.: PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. pp. 9895–9901. Association for Computational Linguistics (Nov 2021), `https://aclanthology.org/2021.emnlp-main.779`
10. Shaw, P., Chang, M.W., Pasupat, P., Toutanova, K.: Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). pp. 922–938. Association for Computational Linguistics, Online (Aug 2021). https://doi.org/10.18653/v1/2021.acl-long.75, `https://aclanthology.org/2021.acl-long.75`

11. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. Advances in neural information processing systems **28** (2015)
12. Wang, B., Shin, R., Liu, X., Polozov, O., Richardson, M.: Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. arXiv preprint arXiv:1911.04942 (2019)
13. Xie, T., Wu, C.H., Shi, P., Zhong, R., Scholak, T., Yasunaga, M., Wu, C.S., Zhong, M., Yin, P., Wang, S.I., Zhong, V., Wang, B., Li, C., Boyle, C., Ni, A., Yao, Z., Radev, D., Xiong, C., Kong, L., Zhang, R., Smith, N.A., Zettlemoyer, L., Yu, T.: Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. EMNLP (2022)
14. Yu, T., Zhang, R., Er, H.Y., Li, S., Xue, E., Pang, B., Lin, X.V., Tan, Y.C., Shi, T., Li, Z., et al.: Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. arXiv preprint arXiv:1909.05378 (2019)
15. Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al.: Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. arXiv preprint arXiv:1809.08887 (2018)
16. Zhong, V., Xiong, C., Socher, R.: Seq2sql: Generating structured queries from natural language using reinforcement learning. CoRR **abs/1709.00103** (2017)