Increasing Accuracy of LLM-powered Question Answering on SQL databases: Knowledge Graphs to the Rescue

Juan Sequeda, Dean Allemang, Bryon Jacob
data.world AI Lab
{juan,dean.allemang,bryon}@data.world

Abstract

Large Language Models (LLMs) are promising technology to support Question Answering on enterprise SQL data (i.e. Text-to-SQL). Knowledge Graphs are also promising technology to enhance LLM-based question answering by providing business context that LLMs lack. However, it is not well understood to what extent Knowledge Graphs can increase the accuracy of LLM-powered question answering system on SQL databases. Our research aims to understand and quantify this extent. First, we introduce a benchmark comprising an enterprise SQL schema in the insurance domain, a range of enterprise queries encompassing reporting to metrics, and a contextual layer consisting of an ontology and mappings that define a Knowledge Graph. The experimental reveals that question answering using GPT-4, with zero-shot prompts directly on SQL databases, achieves an accuracy of 16%. Notably, this accuracy increases to 54% when questions are posed over a Knowledge Graph representation of the enterprise SQL database. Second, we present an approach that leverages the ontology of the Knowledge Graph to deterministically detect incorrect queries generated by the LLM and repair them. Experimental results show that the accuracy increases to 72.55%, including an additional 8% of *"I don't know"* unknown results. Thus, the overall error rate is 20%. The conclusion is that investing in Knowledge Graph provides higher accuracy for LLM powered question answering systems on SQL databases.

A Introduction

Business users and executives would like to have an AI assistant that understands their business, available to them at all times, in order to ask questions and receive accurate, explainable and governed answers. This challenge, known as Question Answering, which is the ability to interact with data using natural language questions and obtaining accurate results, has been a long-standing challenge in computer science dating back to the 1960s [12–14, 28]. The field has advanced throughout the past decades [6, 27, 31], through Text-to-SQL approaches, as a means of facilitating chatting with the data that is stored in SQL databases[9, 17, 21, 24, 29, 33]. With the rise of Generative AI and Large Language Models (LLMs) in early 2023, the interest increased dramatically. These question answering systems hold tremendous potential for transforming the way data-driven decision making is executed within enterprises.

Knowledge Graphs (KGs) have been identified as a promising solution to fill the business context gaps in order to reduce hallucinations, thus enhancing the accuracy of LLMs. The effective integration of LLMs and KGs started to gaining traction in academia in the past several years[1][23]. From an industry perspective, Gartner stated in July 2023 that, *"Knowledge graphs provide the perfect complement to*

[1]https://github.com/RManLuo/Awesome-LLM-KG

LLM-based solutions where high thresholds of accuracy and correctness need to be attained."[2].

Our hypothesis is that Knowledge Graphs play a critical role in LLM powered Question Answering systems on SQL databases. However, at the time that we started this research in July 2023, it was not clear to what extent. The starting point of our work is to understand the role of Knowledge Graphs for accuracy, given that hallucinations became one of the largest concerns in the industry. Our work comes in two parts.

First, we seek to understand the accuracy of LLM-powered question answering systems with respect to enterprise questions, enterprise SQL databases and the role knowledge graphs play to improve the accuracy. Our first contribution [25, 26] is a benchmark with experimental results showing that by using GPT-4 and zero-shot prompting, enterprise natural language questions over enterprise SQL databases schema and generating a SQL query achieved 16.7% accuracy. This accuracy increased to 54.2% when a SPARQL query was evaluated over a Knowledge Graph representation of the SQL database in the form of an OWL ontology and R2RML mapping, thus an accuracy improvement of 37.5%. The benchmark can be found here: `https://github.com/datadotworld/cwd-benchmark-data`. This contribution has made an impact in the industry. The benchmark and the results were initially independently reproduced and validated by dbt Labs[3]. Several semantic layer vendors have further validated our results[4] [5] [6] [7] [8] [9]. The GraphRAG Manifesto by Neo4j argues that one of the benefits of GraphRAG relative to vector-only RAG is due to higher accurate responses, citing our benchmark and results[10].

Leveraging the learnings from our first contribution, namely understanding what happened with inaccurate queries, our intuition is that accuracy can be further increased by 1) leveraging the ontology of the knowledge graph to check for errors in the LLM generated SPARQL queries and 2) using the LLM to repair incorrect queries. Our second contribution [3, 4] is a two-part approach consisting 1) Ontology-based Query Check (OBQC), which checks in a deterministic manner if the query is valid by applying rules based on the semantics of the ontology. If the OBQC detects an error, we could either determine to not return the result thus terminate or we could try to repair the query, and 2) LLM Repair, which repairs the detected incorrect SPARQL query generated by the LLM. The result is a new query which can then be passed back to the OBQC. By grouping all the questions in the benchmark, the OBQC and LLM Repair increased the accuracy 72.55%. If the repairs were not successful after three iterations, an unknown result was returned, which occurred 8% of the time. The result is an error rate of 20%.

The conclusion of our work is that Knowledge Graph provides higher accuracy for LLM powered question answering systems on SQL databases. Therefore, enterprises that are considering to use LLMs for question answering on their SQL databases must invest in knowledge graphs.

[2]Adopt a Data Semantics Approach to Drive Business Value," Gartner Report by Guido De Simoni, Robert Thanaraj, Henry Cook, July 28, 2023

[3]https://roundup.getdbt.com/p/semantic-layer-as-the-data-interface

[4]https://www.atscale.com/blog/semantic-layers-make-genai-more-accurate/

[5]https://www.wisecube.ai/blog/optimizing-llm-precision-with-knowledge-graph-based-natural-language-qa-systems/

[6]https://blog.kuzudb.com/post/llms-graphs-part-1/

[7]https://delphihq.substack.com/p/delphi-at-100-dbt-semantic-layer

[8]https://cube.dev/blog/semantic-layers-the-missing-piece-for-ai-enabled-analytics

[9]https://www.stratio.com/blog/stratio-business-semantic-data-layer-delivers-99-answer-accuracy-for-llms/

[10]https://neo4j.com/blog/graphrag-manifesto/

B Understanding the role of Knowledge Graphs on LLM's Accuracy for Question Answering on SQL

While question answering systems have shown remarkable performance in several Text-to-SQL benchmarks [7, 8], such as Spider [30], WikiSQL[33], KaggleDBQA[19] their implications relating to enterprise SQL databases remain relatively obscure. We argue that existing Question Answering and Text-to-SQL benchmarks, although valuable, are often misaligned with real-world enterprise settings:

1. these benchmarks typically overlook complex database schemas representing enterprise domains, which likely comprise hundreds of tables,

2. they also often disregard questions that are crucial for operational and strategic planning in an enterprise, including questions related to business reporting, metrics, and key performance indicators (KPIs), and

3. a critical missing link is the absence of a business context layer – metadata, mappings, transformations, ontologies, that provides business semantics and knowledge about the enterprise.

Recent benchmarks [20, 22] are attempting to address the first challenge. However, the second and specially the third point have not been a focus of those new benchmarks. Without these vital components, LLMs for enterprise question answering on SQL databases risk being disconnected from the reality of enterprise data, leading to hallucinations and uncontrolled outcomes.

We investigate the following two research questions:

RQ1: To what extent Large Language Models (LLMs) can accurately answer enterprise natural language questions over enterprise SQL databases.

RQ2: To what extent Knowledge Graphs can improve the accuracy of Large Language Models (LLMs) to answer enterprise natural language questions over enterprise SQL databases.

The hypothesis is the following: *An LLM powered question answering system that answers a natural language question over a knowledge graph representation of the SQL database returns more accurate results than an LLM powered question answering system that answers a natural language question over the SQL database without a knowledge graph.*

Enterprise SQL Schema The enterprise SQL schema used in the benchmark comes from the P&C Data Model for Property And Casualty Insurance[11], a standard model created by Object Management Group (OMG), a standards development organization. This OMG specification addresses the data management needs of the Property and Casualty insurance community.

Enterprise Questions The benchmark comes with 43 Question-Answer pairs as evaluation criteria, where the input is the question, and the output is the corresponding answer to the question based on a data instance. The questions are written in English, and refer to concepts covered by the data. Since there can be multiple valid SQL queries for a given question, the determining accuracy factor is the final output instead of a generated SQL query. In order to score the execution accuracy of an LLM, we need to have a reference answer to each question. An "answer" in this situation is itself a query; it is a query that was written by a human expert, which gives the expected correct answer to the question. Each question has a reference query in SQL for the relational database, and SPARQL for the knowledge graph. Naturally, each query gives the same response when run against the data.

The questions are classified on a spectrum of low to high complexity:

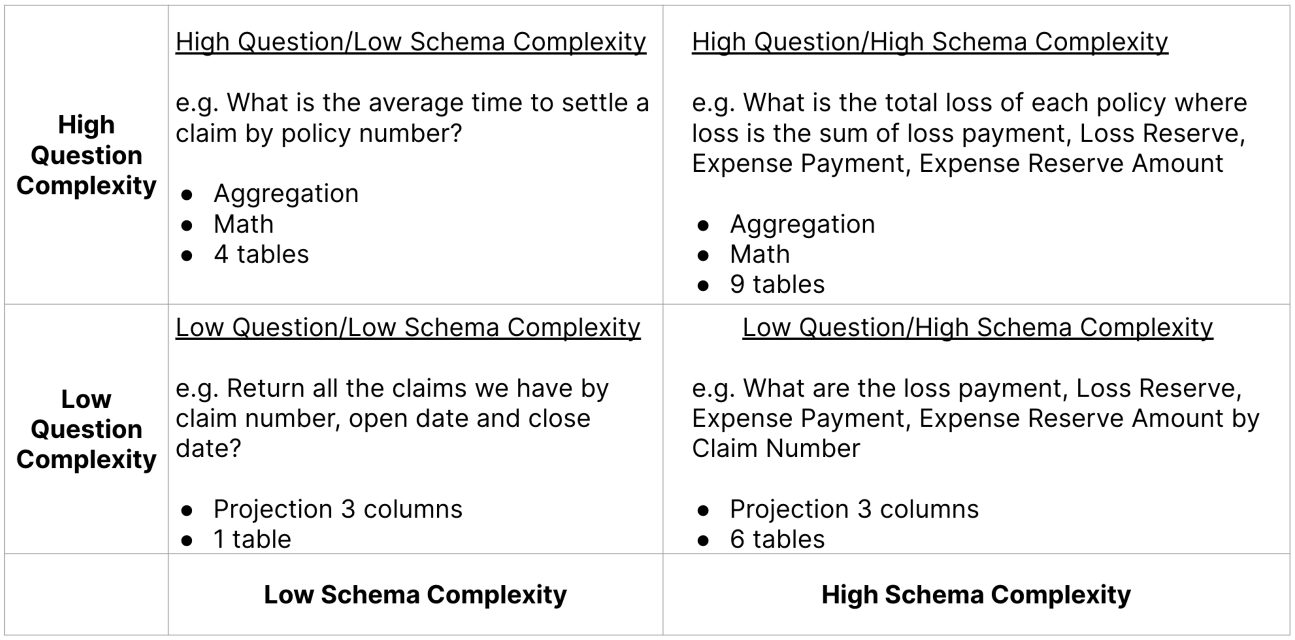
[11] `https://www.omg.org/spec/PC/1.0/About-PC`

- Low question complexity: Pertains to business reporting use cases, aimed at facilitating daily business operations. From a technical standpoint, these questions are translated into SELECT-FROM SQL queries.
- High question complexity: Arises in the context of Metrics and Key Performance Indicators (KPIs) within an organization. These questions are posed to make informed strategic decisions crucial for organizational success. From a technical standpoint, these questions are translated to SQL queries involving aggregations and mathematical functions.

Questions also depend on the number of tables required to provide an answer. Therefore, questions are also classified on a spectrum of low to high schema:

- Low schema complexity: Small number of tables (i.e. 0 - 4), denormalized schema
- High schema complexity: Larger number of tables (5+), normalized schema, many-to many join tables, etc.

By combining these two spectrums, four quadrants are defined which are used to classify the questions as shown in Figure 32:



High Question Complexity	High Question/Low Schema Complexity e.g. What is the average time to settle a claim by policy number? ● Aggregation ● Math ● 4 tables	High Question/High Schema Complexity e.g. What is the total loss of each policy where loss is the sum of loss payment, Loss Reserve, Expense Payment, Expense Reserve Amount ● Aggregation ● Math ● 9 tables
Low Question Complexity	Low Question/Low Schema Complexity e.g. Return all the claims we have by claim number, open date and close date? ● Projection 3 columns ● 1 table	Low Question/High Schema Complexity e.g. What are the loss payment, Loss Reserve, Expense Payment, Expense Reserve Amount by Claim Number ● Projection 3 columns ● 6 tables
	Low Schema Complexity	**High Schema Complexity**

Figure 32: Four quadrants to classify questions: (1) Low Question/Low Schema Complexity, (2) High Question/Low Schema Complexity, (3) Low Question/High Schema Complexity, and (4) High Question/High Schema Complexity

This 43 questions of the benchmark can be found in [25] and on Github[12]. While 43 questions may be considered small, the benchmark ensures coverage of key scenarios that reflect real-world enterprise data usage and queries in the insurance domain. All the questions in the benchmark are building blocks to answer one of the most important key metrics in the insurance industry: Loss Ratio. While benchmarks with larger numbers of questions can provide generalizability to evaluate question answering systems and setup leaderboards, the goal of this benchmark is to understand the role of Knowledge Graphs and to

[12]`https://github.com/datadotworld/cwd-benchmark-data`

what extent the accuracy improves. The quadrant provides visibility on the type of extent. Furthemore, the question quadrant can be considered as a framework to be applied for other domains; instead of generating a *laundry list* of questions, categorize them in these quadrants.

Context Layer The context layer consists of two parts:

- Ontology: Business Concepts, Attributes, and Relationships that describe the insurance domain.
- Mapping: transformation rules from the source SQL schema to the corresponding Business Concepts, Attributes, and Relationships in the target ontology.

For this current version of the benchmark, the context layer is provided in machine readable as RDF: ontology in OWL and mapping in R2RML. The OWL ontology and R2RML mappings can be used to create the Knowledge Graph either in a virtualized or materialized way.

Scoring The benchmark reports three scores: Execution Accuracy, Overall Execution Accuracy and Average Overall Execution Accuracy.

- Execution Accuracy (EA): We follow the metric of Execution Accuracy (EA) from the Spider benchmark [30]. An execution is accurate if the result of the query matches the answer for the query. Note that the order or the labels of the columns are not taken in account for accuracy.
- Overall Execution Accuracy (OEA): Given the non-deterministic nature of LLMs, there is no guarantee that given an input question, the generated query will always be the same thus providing the same answer. Therefore, every question has a Overall Execution Accuracy (OEA) score which is calculated as (# of EA)/Total Number of runs.
- Average Overall Execution Accuracy (AOEA): The Average Overall Execution Accuracy is the average number of OEA scores for a given set of questions. This set could be for all the questions in the benchmark or all the questions in a quadrant.

The benchmark serves as a framework for the results to be reproduced in an enterprise's own setting using their own enterprise schemas, questions and context.

B.1 Experimental Setup

The question answering system we evaluated was a zero-shot prompt to GPT-4, that is instructed to generate a query, which is executed against the database. The resulting response is compared to the response given by the reference query.

The particular parameters to the OpenAI API are as follows:

- max_tokens = 2048
- n = 1
- temperature = 0.3

Additionally, a timeout was set so that computations that take more than 60 seconds are considered to be failures.

Note that the goal of our experiment is to understand the role of Knowledge Graphs on accuracy. The focus of the experiment is not to understand how a certain LLM performs with a Knowledge Graph. That is why we select only one LLM for our experiment, namely GPT-4. Naturally, future work should include a comparison of multiple LLMs in order to understand how a Knowledge Graph can increase accuracy on different LLMs.

B.2 Question Answering System for SQL

The question answering system for SQL is shown in Figure 33. The question and the SQL DDL for the database are provided as zero-shot prompt to GPT-4. These are combined together using the following simple prompt template:

SQL Zero-shot Prompt

```
INSERT SQL DDL
Write a SQL query that answers the following question. Do not explain
the query. Return just the query, so it can be run verbatim from your
response.
Here's the question:
INSERT QUESTION
```

We kept the prompt simple for this experiment, because we wanted to focus on the ability of the contextual information (the DDL in the case of SQL) to provide necessary information for the formation of the query.

The resulting query is sent verbatim to the SQL processor of data.world, which returns an answer in a tabular form. This is converted into a Pandas DataFrame for comparison. At the same time, the reference query for the question is sent to data.world, and its result is also converted to a DataFrame. Once they are both in the form of DataFrames, it is a simple matter to compare them. Details of this comparison are available from the Spider project[30].

B.3 Question Answering System for Knowledge Graph

The question answering system for the Knowledge Graph is shown in Figure 34. The question and the OWL ontology are provided as zero-shot prompt to GPT-4. These are combined together using the following simple prompt template:

SPARQL Zero-shot Prompt

As in the SQL case, we kept the prompt simple. The extra line about the SERVICE allows the LLM to produce queries that invoke the data.world knowledge graph virtualization layer. In principle, this adds some complexity to the SPARQL prompt, but in practice, GPT-4 seemed to handle it very well.

The resulting query is sent verbatim to the SPARQL processor of data.world, and the result converted to a DataFrame, just as for the SQL case.

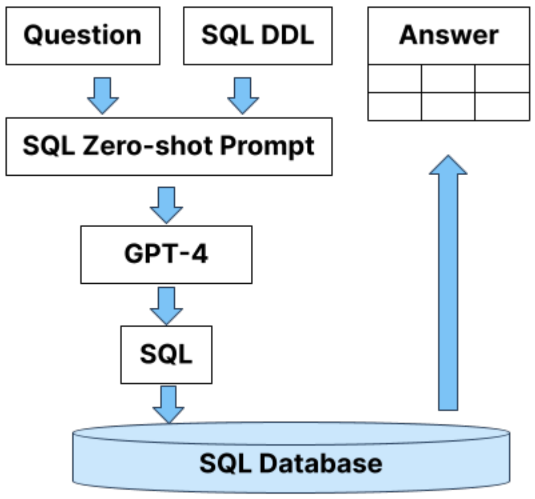


Figure 33: Question Answering System for SQL

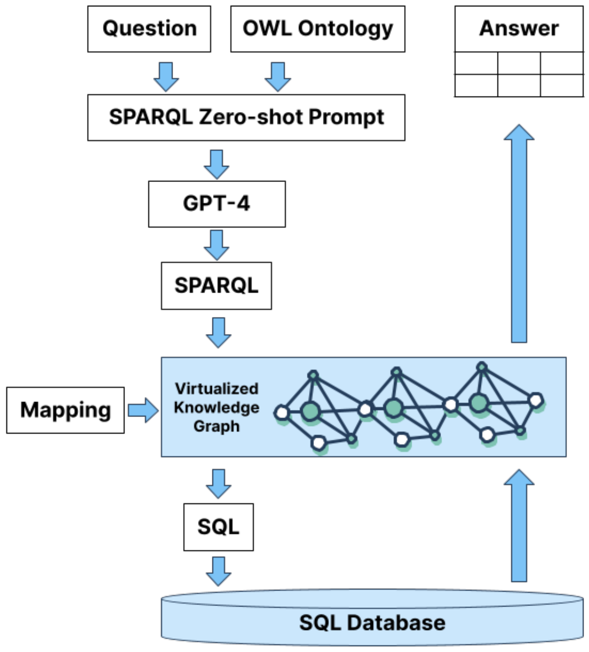


Figure 34: Question Answering System for Knowledge Graph

B.4 Results

The results are presented in four parts 1) overall, 2) question quadrant, 3) partial accuracy and 4) inaccurate results. In the results, we refer to

- SPARQL as question over Knowledge Graph representation of the SQL database and
- SQL as questions directly on the SQL databases without a Knowledge Graph.

Given that the OEA of a question is a percentage, the results are presented as a heatmap. Every cell corresponds to a generated query for the given question. The value in the cell is the OEA for that question. The green color corresponds to 100% OEA. The red color corresponds to 0% OEA. The color scale goes from green to red.

The Overall and Quadrant results are presented in Table 18.

B.4.1 Overall

By grouping all the questions in the benchmark, SQL achieves an AOEA of 16.7%. In comparison, SPARQL achieves an average OEA of 54.2% as shown in Figure 35. The heatmap that depicts the OEA for each question is shown in Figure 36: Therefore, overall SPARQL accuracy was 3x the SQL accuracy.

Overall, Natural Language questions translated to SPARQL over a Knowledge Graph representation of the SQL database achieved 3x the accuracy of natural language questions translated to SQL and executed directly over the SQL database. Combining all the questions into one overall result is not satisfactory because there are nuances to the types of questions. This is why we also present the results in each of the quadrants.

B.4.2 Quadrant

Figure 37 presents the AOEA scores for questions in each quadrant. Figure 38 presents the heat map for each quadrant. We observe the following results:

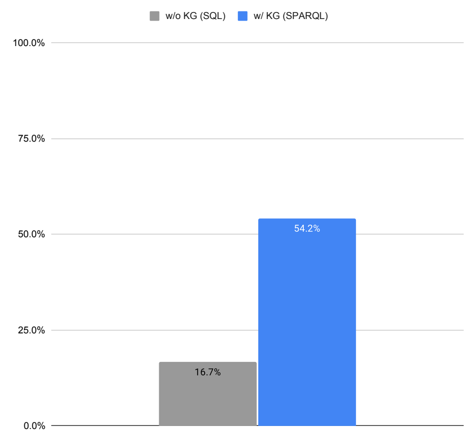


Figure 35: Average Overall Execution Accuracy (AOEA) of SPARQL and SQL for all the questions in the benchmark

- Low Question/Low Schema: SQL achieves an AOEA of 25.5%. In comparison, SPARQL achieves an AOEA of 71.1%. The SPARQL accuracy is 2.8X the SQL accuracy.
- High Question/Low Schema: SQL achieves an AOEA of 37.4%. In comparison, SPARQL achieves an AOEA of 66.9%. The SPARQL accuracy is 1.8X the SQL accuracy.
- Low Question/High Schema: SQL was not able to answer any question accurately. In comparison, SPARQL achieves an AOEA of 35.7%.
- High Question/High Schema: SQL was not able to answer any question accurately. In comparison, SPARQL achieves an AOEA of 38.7%.

Per the hypothesis, SPARQL achieves higher accuracy than SQL in every quadrant. Furthermore, it is surprising to observe that SQL was not able to answer any question in the High Schema Complexity quadrants. These results by quadrant sheds further light on understanding the extent. In each quadrant, SPARQL accuracy is higher than the SQL accuracy. While the SPARQL accuracy is 2.8X the SQL accuracy for Low Question/Low Schema and 1.8X for High Question/Low Schema, it was unforeseen that SQL was not able to accurately answer any questions for Low Question/High Schema and High Question/High Schema. The results also lead us to understand when SQL starts to fail. When a question requires more than 4 tables to provide then answer, the accuracy drops to zero.

	w/o KG (SQL)	**w/ KG (SPARQL)**	**Improvement**
All Questions	16.7%	54.2%	37.5%
Low Question/Low Schema	25.5%	71.1%	45.6%
High Question/Low Schema	37.4%	66.9%	29.5%
Low Question/High Schema	0%	35.7%	35.7%
High Question/High Schema	0%	38.5%	38.5%

Table 18: Average Overall Execution Accuracy (AOEA) of Overall and Quadrant Results

B.4.3 Partial Accuracy

We manually analyzed the generated SQL and SPARQL queries and observed that a subset of queries produced partially accurate results. We consider a partially accurate answer to be one where the returned answers are accurate but incomplete. During the manual analysis, the following patterns for partially accurate answers are observed:

- **Overlap:** the columns returned by the query are correct, however, they are a subset of the accurate answer. In some cases, they include other columns that are not part of the expected answer. This can be seen as a form of a semantic overlap[10].
- **Return Identifier:** An internal identifier was returned instead of the appropriate label.

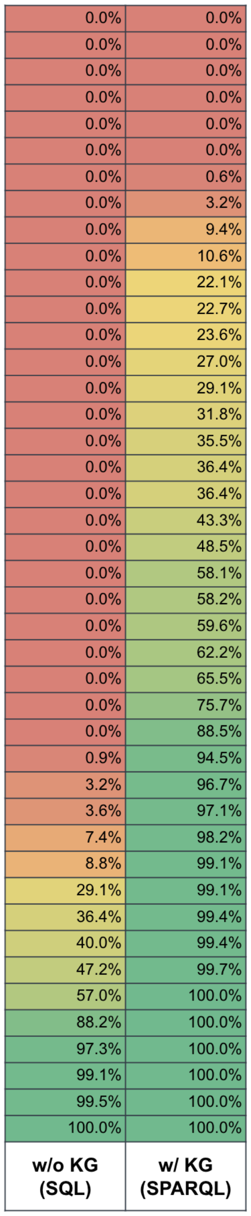
Consider the question *Return all the claims we have by claim number, open date and close date?* and the following generated SQL and SPARQL query:

SQL

```
SELECT Claim_Identifier, Claim_Open_Date, Claim_Close_Date
FROM Claim
```

SPARQL

```
SELECT ?claim ?claimOpenDate ?claimCloseDate
WHERE {
    ?claim a in:Claim ;
           in:claimNumber ?claimNumber ;
           in:claimOpenDate ?claimOpenDate ;
           in:claimCloseDate ?claimCloseDate .
}
```



w/o KG (SQL)	w/ KG (SPARQL)
0.0%	0.0%
0.0%	0.0%
0.0%	0.0%
0.0%	0.0%
0.0%	0.0%
0.0%	0.0%
0.0%	0.6%
0.0%	3.2%
0.0%	9.4%
0.0%	10.6%
0.0%	22.1%
0.0%	22.7%
0.0%	23.6%
0.0%	27.0%
0.0%	29.1%
0.0%	31.8%
0.0%	35.5%
0.0%	36.4%
0.0%	36.4%
0.0%	43.3%
0.0%	48.5%
0.0%	58.1%
0.0%	58.2%
0.0%	59.6%
0.0%	62.2%
0.0%	65.5%
0.0%	75.7%
0.0%	88.5%
0.9%	94.5%
3.2%	96.7%
3.6%	97.1%
7.4%	98.2%
8.8%	99.1%
29.1%	99.1%
36.4%	99.4%
40.0%	99.4%
47.2%	99.7%
57.0%	100.0%
88.2%	100.0%
97.3%	100.0%
99.1%	100.0%
99.5%	100.0%
100.0%	100.0%

Figure 36: Overall Execution Accuracy (OEA) of SPARQL and SQL for all the questions as a heatmap

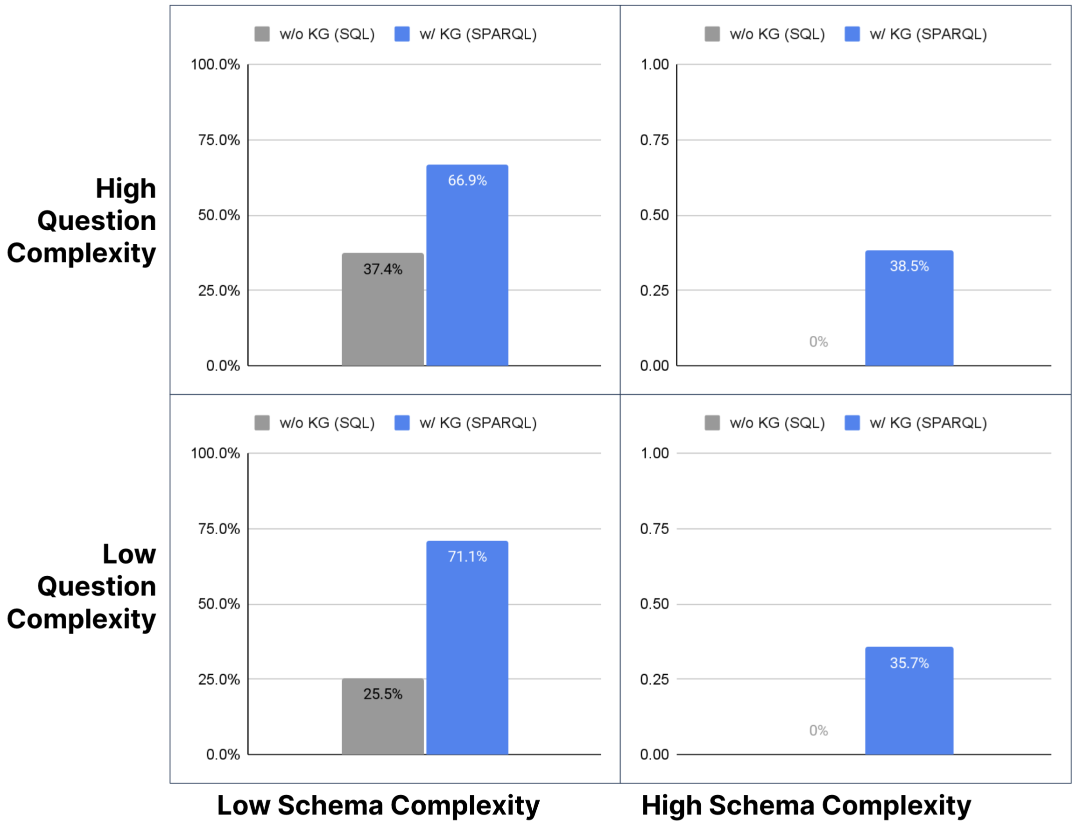


Figure 37: Average Overall Execution Accuracy (AOEA) of SPARQL and SQL for each quadrant

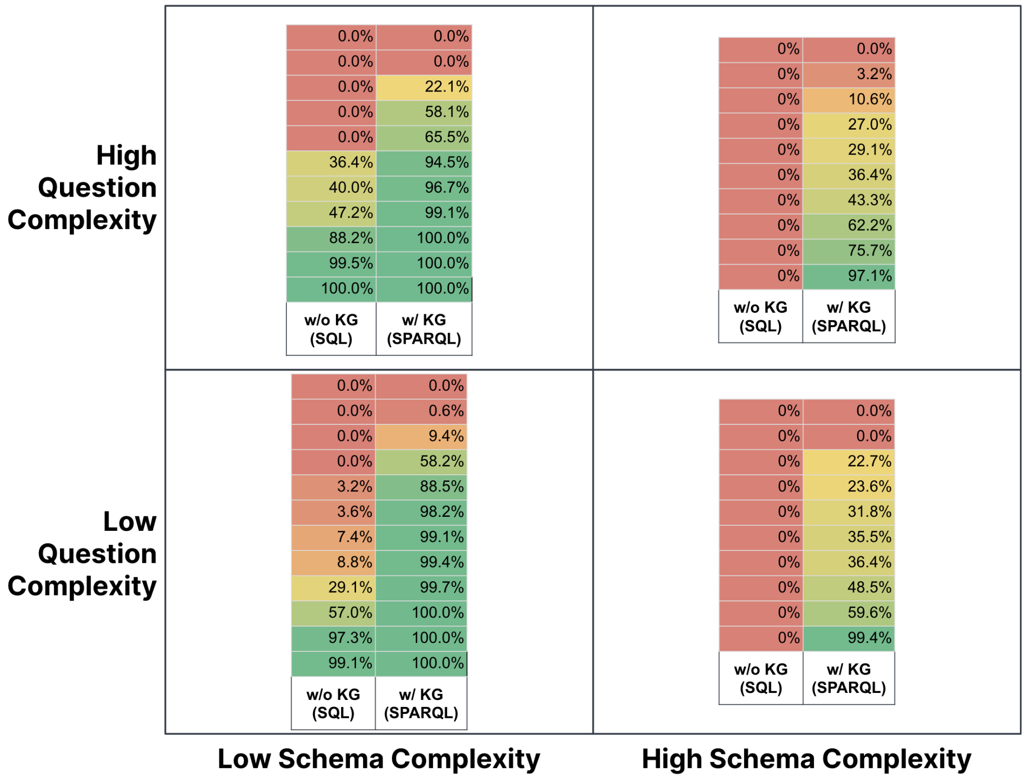


Figure 38: Overall Execution Accuracy (OEA) of SPARQL and SQL for each quadrant as a heatmap

The answer for claim open date and claim close date are accurate and is a subset of the correct answer. However in the SQL query, the `Claim_Identifier` column is being returned as the claim number, when in fact, the claim number is actually the column `company_claim_number`. In the SPARQL query case, the variable `?claim` is returned which binds to the IRI that uniquely identifies each claim. The claim number is not returned.

In practice, if a user is interacting with a system and the results are missing a column, they could ask for the missing column or provide a label instead of an identifier. Therefore partial accuracy may be acceptable for users. However this is an open question on how to define partial accuracy and how to score it.

B.4.4 Inaccuracy

During the manual analysis of the generated queries, we also observed query characteristics that generated the inaccurate answers. These characteristics were different for SQL and SPARQL.

SQL Inaccuracy The following three types of inaccuracies were observed:

- **Column Name Hallucinations**: Column names were generated that do not exist in the corresponding table.
- **Value Hallucinations**: Generated value applied as a filter on a column where that value does not exist in the database.
- **Join Hallucinations**: Generated joins that are not accurate.

SPARQL Inaccuracy

- **Incorrect Path:** The generated query does not follow the correct path of the properties in the ontology. The generated path goes from A to C when the correct path is A to B to C.
- **Incorrect Direction:** The generated query swaps the direction of a property. The generated direction is B to A, when the correct direction is A to B.

The inaccuracy of SQL queries are based on hallucination while the inaccuracy of SPARQL queries are based on path inconsistency. The SQL hallucinations are evident: column names that don't exist in a table and values that the LLM does not know if they exist in the data. The joins may seem plausible, but they are not how the database was designed, thus returning empty results. For SPARQL queries, the generated paths are indicative that the LLM knew what the correct starting and end node was and the error was on defining the correct path from the start node to the end node. One could even argue that the LLM appears to do some sort of reasoning but not always getting it correct. This observation is what led us to the second part of our work.

C Ontologies to the Rescue

Consider the following knowledge represented in an ontology of a knowledge graph: *a Policy is sold by an Agent*. If an LLM generated SPARQL query representing the statement *an Agent is sold by a Policy*, it would be inconsistent because it does not match the semantics of the ontology (i.e. this doesn't make sense). Our intuition is two-fold. First, by leveraging the ontology of knowledge graph, we can check the LLM generated SPARQL query and detect these types of errors. Second, we can also use the LLM to repair incorrect SPARQL queries.

For example, assume the following question *"return all the policies that an agent sold"*, resulted in the following SPARQL query:

```
SELECT ?agent ?policy
WHERE {
  ?agent :soldByAgent ?policy .
  ?agent rdf:type :Agent
}
```

and given the following snippet of an OWL ontology

```
:soldByAgent a owl:DatatypeProperty;
  rdfs:domain :Policy ;
  rdfs:range :Agent .
```

we could determine that the generated query should be correct if the domain of `:soldByAgent` is `:Policy`. However, per the query, the domain is `:Agent` and assuming they are disjoint, the generated query does not match the semantics of the ontology, thus it is incorrect. Given an explaination of this error, we could then prompt the LLM to try again.

We investigate the following research questions:
RQ3: To what extent can the accuracy increase by leveraging the ontology of a knowledge graph to detect errors of a SPARQL query and an LLM to repair the errors?
RQ4: What types of errors are most commonly presented in SPARQL queries generated by an LLM?

The hypothesis is the following: *An ontology can increase the accuracy of an LLM powered question answering system that answers a natural language question over a knowledge graph.*

This first part of our approach is the **Ontology-based Query Check** (OBQC), which checks if the query is valid by applying rules based on the semantics of the ontology. A set of rules checks the semantics of the body of the query (i.e. the WHERE clause). Another set of rules checks the head of query (i.e. the SELECT clause). It is important to clarify that the Ontology-based Query Check does not use an LLM. It is a deterministic rule-based approach based solely on the semantics of the ontology. If the OBQC detects an error, we could try to repair the query.

Repairing databases[1] and programs[11, 32] has been an long standing research area in computer science. Recently, LLMs have been applied to repair programs[5, 18]. Inspired by these approaches, consider the following example. Once a SPARQL query is detected to be incorrect, we can define an explanation for the reason why it is incorrect. Per our running example, an explanation is the following: *The property :soldByAgent has domain :Policy, but its subject ?agent is a :Agent, which isn't a subclass of :Policy.*. What if we can pass the incorrect SPARQL query, with this explanation and prompt the LLM to rewrite the query?

The second part of our approach is **LLM Repair**, which repairs the SPARQL query generated by the LLM. It takes as input the incorrect query and the explanation coming from the rule(s) that was fired as an explanation to why the query is incorrect and re-prompts the LLM. The result is a new query which can then be passed back to the OBQC. Our approach gives us the opportunity to understand the capability of an LLM to repair a SPARQL query and thus further improve the accuracy. Figure 39 depicts the overview of our approach.

C.1 Ontology-based Query Check

Knowledge Graphs defined using the Semantic Web technology stack (specifically, RDF, RDFS, OWL and SPARQL) have been built on a rigorous logical foundation. The exact meaning of a statement (triple) in RDF is given in terms of predicate logic; the meaning of a model in RDFS or OWL is specified according to a logical foundation [2, 15, 16][13]. The meaning of a SPARQL query is specified in terms of these logical foundations. A practical upshot of this theoretical framework is that it is possible to know exactly what constraints a model in RDFS or OWL places on the correctness of a SPARQL query, and these constraints can be described in an executable way in SPARQL. The approach takes two inputs: a SPARQL query and an ontology. The output consists of a list of sentences that describe ways in which the SPARQL query deviates from the specifications in the ontology.

The check system relies on the declarative nature of SPARQL, the structure of Basic Graph Patterns and, the ability to query the ontology via SPARQL itself. If the generated query deviates from the ontology, the approach outlines how. The approach to achieve this is threefold:

First, a SPARQL query consists of a pattern to be matched against the data (specified after the keyword WHERE in the query); known as a Basic Graph Pattern (BGP) of the query. The process begins with extraction of BGPs from the generated SPARQL query, replacing variables with resources from a reserved namespace (prefixed with qq:). Some portions of the original query logic, including the SELECT clause, subquery structures, filters, UNIONs, OPTIONAL and NOT clauses, aren't considered since the focus is on examining the compatibility of the BGP with the ontology structure. We leave that for future work. However, note that violations of BGPs in an OPTIONAL or FILTER NOT EXISTS / MINUS context are not ignored as they can also provide vital insights into the query understanding.

Second, a *conjunctive graph*[14] is constructed by encapsulating two named graphs: `:query` and `:ontology`, representing the SPARQL query's BGP-turned-RDF and the ontology, respectively.

[13]For an introduction to knowledge graphs and semantic web technologies, we refer the reader to the textbooks "Semantic Web for the Working Ontologist" and "Knowledge Graphs" `https://kgbook.org/`

[14]using RDFLib nomenclature

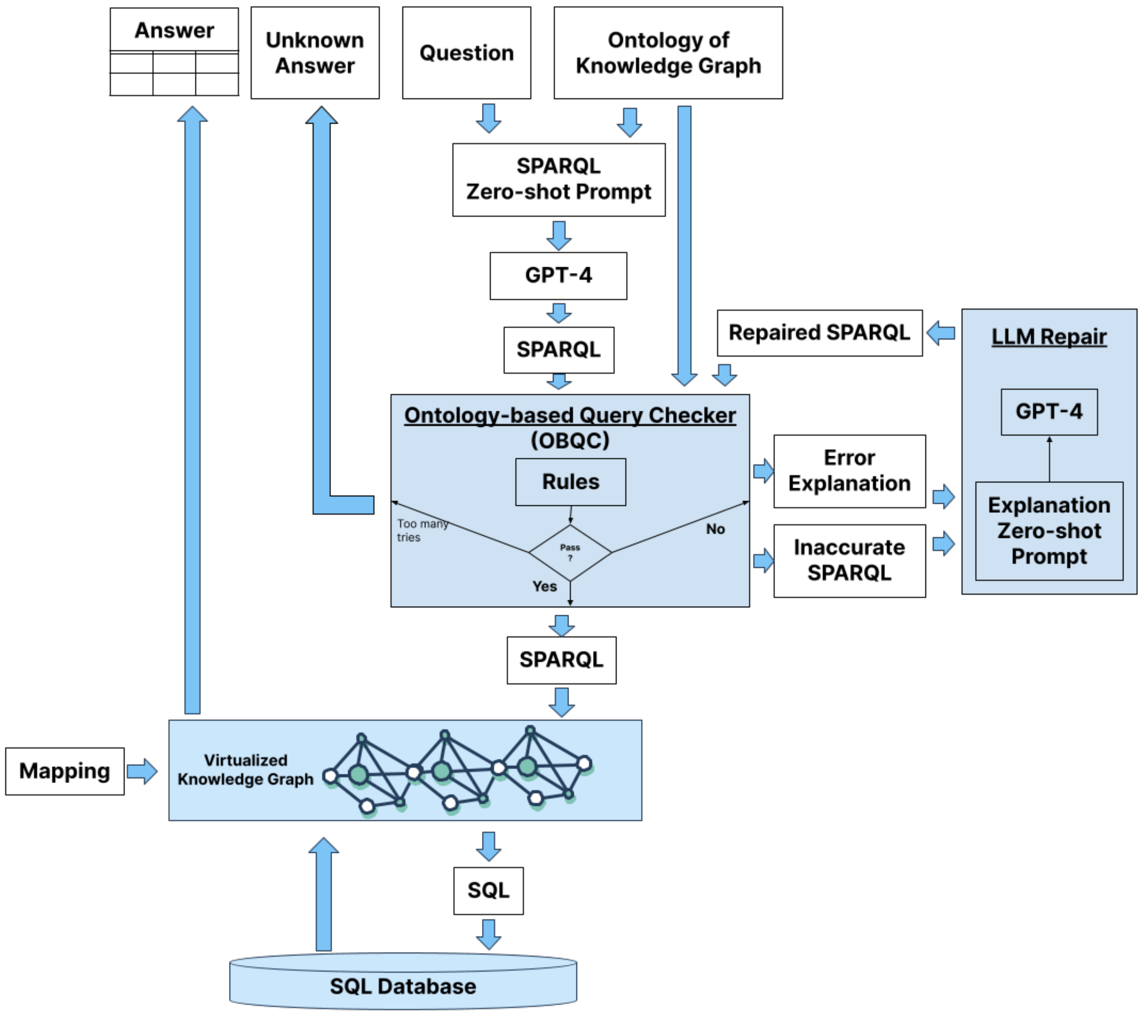


Figure 39: Overview of our Ontology-based Query Checker and LLM Repair approach

Third, the ontology consistency rules are applied guided by the formal logic of RDFS and OWL. The rules are implemented in SPARQL which query the `:query` and `:ontology` graphs and identify instances where the query diverges from the ontology.

Note that this ontology-based query check approach could be extended to other Knowledge Graphs that lack a rigorous semantic foundation, such as property graphs. However, property graphs do not have well-defined standardized schema language and an inference system. This would need to be explicitly defined in order for our approach to be applied.

C.1.1 Rules

We defined two set of rules based on the body and the head of a query. The body rules check that the Basic Graph Patterns of a SPARQL query (i.e. WHERE clause) matches the semantics of an ontology. Our approach follows a subset of the RDF Schema (RDFS) semantics[15]. The body rules are:

- Domain: If the domain of a property p is a class C, then the subject of any triple using p as a predicate must be a member of class C.
- Range: If the range of a property p is a class C, then the object of any triple using p as a predicate must be a member of class C.
- Double Range: If two triples make conflicting requirements on the range of a property then error.
- Double Domain: If two triples make conflicting requirements on the domain of a property then error
- Domain Range: If the object of a first triple is the subject of a second triple, then the range of the property of the first triple should be the same as the domain of the property of the second triple.
- Incorrect Property: All the properties in the query need to exist in the ontology.

The head rules check the head of a SPARQL query (i.e. SELECT clause). A common error for an LLM is to include extra values in the SELECT clause or to leave some out. These errors have nothing to do with the ontology. However, a very common error is to include a variable in the SELECT clause that will be bound to an IRI (an identifier). The head rules are:

- Subject Output: if a query selects a variable that is the subject of a basic triple pattern, then it is an IRI.
- IRI Output: if a predicate has a specified range which is a class, then the object of that triple is an IRI.

For simplicity, in this paper we only provide an example of the Domain Rule. The description of the implementation for all the rules can be found in [3, 4].

The Domain rule (rdfs:domain in RDF Schema) is defined in English as follows: If the domain of a property p is a class C, then the subject of any triple using p as a predicate must be a member of class C. The domain rule is formally defined as:

```
IF
    ?p rdfs:domain ?C .
    ?s ?p ?o .
THEN
    ?s rdf:type ?C .
```

[15]`https://www.w3.org/TR/rdf11-schema/`

The following SPARQL query is a representation of this domain rule which detects a violation:

```
SELECT ?p ?domain ?s ?class WHERE {
    GRAPH :query{
      ?s ?p ?o .
      ?s a ?class .
    }
    GRAPH :ontology{
      ?p rdfs:domain ?domain.
      FILTER (ISIRI (?domain))
    }
    FILTER NOT EXISTS {
        ?class rdfs:subClassOf* ?domain .
    }
}
```

The following example shows how the domain rule is used to check a BGP against an ontology. Suppose the LLM generated the following query:

```
SELECT ?agent WHERE {
  ?agent :soldByAgent ?policy .
  ?agent rdf:type :Agent
}
```

This query has a BGP consisting of two-triples:

```
 ?agent :soldByAgent ?policy .
 ?agent rdf:type :Agent
```

The BGP is turned into RDF graph by replacing the variables with resources from a reserved namespace (prefixed with qq:):

```
  qq:agent :soldByAgent qq:policy .
  qq:agent rdf:type :Agent
```

Now, suppose the ontology includes the following definition of `:soldByAgent`:

```
:soldByAgent
  rdfs:domain :Policy ;
  rdfs:range :Agent .
```

The conjunctive graph in nquads is the following:

```
:query {
    qq:agent :soldByAgent qq:policy .
    qq:agent rdf:type :Agent  .
 }
:ontology {
    :soldByAgent
      rdfs:domain :Policy ;
      rdfs:range :Agent .
}
```

The first clause, on the graph `:query`, finds the precondition for `rdfs:domain`; there is a triple with predicate `(`?p) whose subject is a member of some `class`. The second clause searches the ontology for a relevant domain definition; that is, that same property `?p` has a specified domain. This query ignores domain definitions that are not IRIs, which would typically include domains that are UNIONs or INTERSECTIONs of other classes. We have simplified the query for exposition in this paper, but it would be easy enough to extend the query to deal with other OWL constructs. We leave this as future work. Finally, the FILTER clause of this query checks to make sure that the class specified in the input query (`?class`) is not included in the domain (`?domain`). If all of these conditions in the evaluation query hold, then we have found a violation of the ontology in the query.

Continuing our example, in the `:query` graph, we have a match for the first clause, with the binding

```
?s -> qq:agent
?p -> :soldByAgent
?class -> :Agent
```

The second clause searches `:ontology` for a triple matching

```
?soldByAgent rdfs:domain ?domain
```

this matches, with `?domain` bound to `:Policy` (which is indeed an IRI). Finally, we test whether

```
:Agent rdfs:subClassOf* :Policy .
```

Notice that the meaning of the * in SPARQL implies that this would succeed if :Agent were the same as :Policy. But in this case, they are not the same, and there is no such triple, so the FILTER NOT EXISTS condition succeeds, and the check comes up with a match, with the following bindings

```
?p -> :soldByAgent
?domain -> :Policy
?s -> qq:agent
?class -> :Agent
```

This information is not very understandable to a human, and might not be usable to an LLM. But it can be formatted into a meaningful sentence in English as follows:

The property `:soldByAgent` has domain `:Policy`, but its subject `?agent` is a `:Agent`, which isn't a subclass of `:Policy`.

For each check rule, we provide a template that can create this explanation. In this case, the template is: *The property {p} has domain {dom}, but its subject {s} is a {class}, which isn't a subclass of {dom}*

C.2 LLM Repair

The LLM Repair is a prompt that takes as two inputs: 1) the list of issues for which the query is incorrect which is the output of the OBQC and 2) the incorrect SPARQL query. The prompt is the following:

Explanation Zero-shot Prompt

The output is a new LLM generated SPARQL query, which is passed again to the OBQC. This cycle repeats until the check pass, or an upper limit of cycles is reached. In our experiments, the limit is 3. In this latter case, the query generation is said to be unknown; there is no point in sending a query that is known to be faulty to the database.

It is instructive to note that the LLM repair focuses on that task; we do not repeat the question nor the ontology to the LLM. The ontology input was taken into consideration by the OBQC, and the question is reflected in the query so far.

It is also noteworthy that there are two possible outcomes; we can achieve a query that we have considerable confidence in (because it matches the semantics of the ontology), or we fail to create such a query. In the latter case, we are aware of the failure of the system. In contrast to a pure LLM-based system which is prone to *hallucinations*, when we get the wrong answer, we know it, and can report that to the user.

C.3 Results

The results consider three cases: 1) accurate queries (that get the right answer), 2) inaccurate queries (that get the wrong answer), and 3) *unknown* queries, queries that we know are incorrect and are not able to repair. As we summarize the results, we follow the metric of Execution Accuracy (EA) from the Spider benchmark [30].

We report the following metrics for a SPARQL query generated by an LLM:

- Execution Accuracy First Time: if the OBQC returns true the first time which results in an accurate execution.
- Execution Accuracy with Repairs: if the OBQC returns false the first time and the LLM Repair results in an accurate execution.
- Execution Unknown with Repairs: if the OBQC returns false the first time and the LLM Repair is unable to repair after three attempts.

C.3.1 Accuracy Results

By grouping all the questions in the benchmark, the Average Overall Execution Accuracy with Repairs is 72.55%. This is an increase of 29.67% based on the Average Overall Execution Accuracy First Time which is 42.88%. The Average Overall Execution Unknown with Repairs is 8% which implies that the LLM Repair is usually able to repair the queries and is still able to identify when queries can not be repaired. By combining the Average Overall Execution Accuracy with Repairs and Average Overall Execution Unknown with Repairs, the error rate is 19.44%. Based on the results shown in Table 19, we observe that the Ontology-based Query Check and LLM Repair favorably increased the accuracy and reduced the error rate in two areas:

- Questions on High Complex Schema: the Ontology-based Query Check and LLM Repair positively impacts the accuracy of all types of questions that are on a high schema complexity.
- Combining Accuracy and Unknowns: “I don’t know" is a valid answer and arguable a better answer than an inaccurate answer. By combining accuracy and unknown, the error rate reduces, notably making a bigger impact in Low Question/Low Schema.

We observe the following details for each quadrant:

- Low Question/Low Schema: the Ontology-based Query Check and LLM Repair increased the accuracy by 25.48%. The Execution Unknown with Repairs was the highest in this quadrant, 12.87%. Combined, this implies that the error rate is 10.46%, the lowest of all the quadrants.

	Average Overall Execution Accuracy First Time	**Average Overall Execution Accuracy with Repairs**	**Average Overall Execution Unknown with Repairs**	**Average Overall Execution Accuracy + Unknown with Repairs**	**Error Rate**
All Questions	42.88%	72.55%	8%	80.56%	19.44%
Low Question / Low Schema	51.19%	76.67%	12.87%	89.54%	10.46%
High Question / Low Schema	69.76%	75.10%	6.02%	81.12%	18.88%
Low Question / High Schema	17.20%	76.33%	3.45%	79.79%	20.21%
High Question / High Schema	28.17%	60.62%	8.40%	69.03%	30.97%

Table 19: Average Overall Execution Accuracy (AOEA) of Overall and Quadrant Results

- High Question/Low Schema: the Ontology-based Query Check and LLM Repair increased the accuracy by 5.34% which was the lowest increase of the quadrants. It is not clear why. The final error rate is 18.88%.
- Low Question/High Schema: the Ontology-based Query Check and LLM Repair had a substantial impact by increasing the accuracy by 59.13% with an error rate of 20.21%.
- High Question/High Schema: the Ontology-based Query Check and LLM Repair had a meaningful impact by increasing the accuracy by 32.45% with an error rate of 30.97%.

Comparing to the results with our first contribution, the accuracy increase is notable. The Average Overall Execution Accuracy of the same zero-shot Text-to-SPARQL prompt on a Knowledge Graph representation of the SQL database, reported in our previous work, was 54.2%, indicating an error rate of 45.8%. With our Ontology-based Query Check and LLM Repair, the error rate is reduced to 19.44%. Figure 40 depicts these results for all the questions in the benchmark. Figure 41 depicts these results for all questions in each quadrant.

A follow up question to understand the *extent* is the following: *How much of the possible execution accuracy improvement was achieved?* In other words, given the number of times the system achieved an accurate answer on the first time, and the total number of runs, we know how much is left for improvement. Therefore, how much of that improvement was achieved? For example, given a total of 10 runs, where 2 of them were accurate on the first try achieving a First Time Execution Accuracy of 20%, means that there are 8 runs left where the OBQC and LLM Repair to repair a query in order to achieve 100% Execution Accuracy with Repairs. Let's say that the system is able to accurately repair 4 times, therefore the Execution Accuracy with Repairs is 60%. The achievable improvement is 50% because the accurate repair occurred 4 times out of the 8 possible times. Achievable Improvement is calculated as

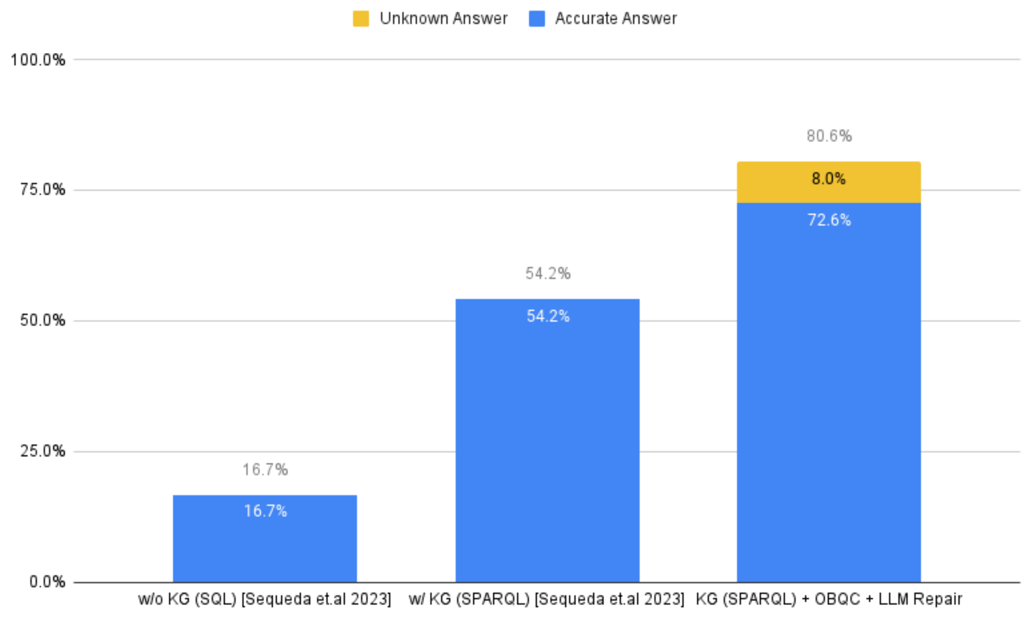


Figure 40: Average Overall Execution Accuracy (AOEA) of SPARQL and SQL for all the questions in the benchmark compared to OBQC and LLM Repair

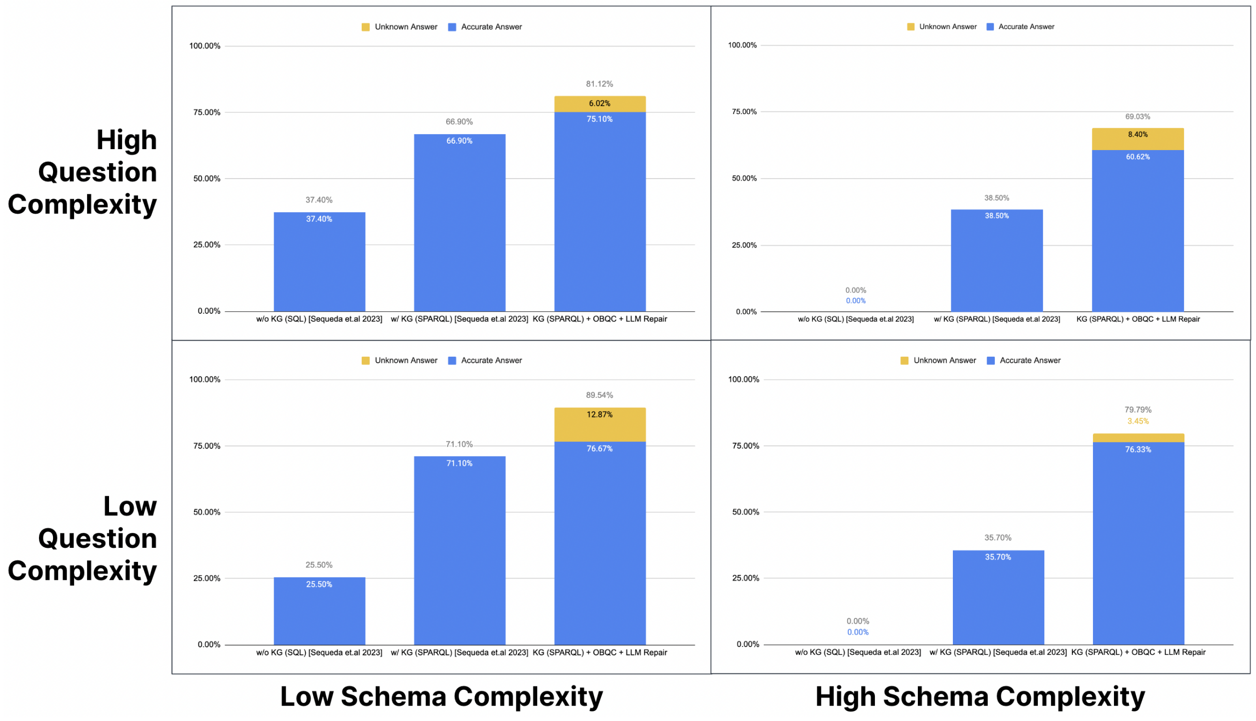


Figure 41: Average Overall Execution Accuracy (AOEA) of SPARQL and SQL for all questions in each quadrant compared to OBQC and LLM Repair

(Number of Accurately Repaired Queries) / (Total Number of runs - Number of First Time Executed Accurate queries). The results of achievable improvement are shown in Table 20

The results indicate that the OBQC is able to successfully repair queries half of the time. Thus, we are halfway there and there is still room for improvement. Recall that OBQC mainly checks the body of the query and just two checks in the SELECT clause determine if IRI identifiers are returned. We postulate that a set of inaccurate queries are due to the overlap type of partial accurate queries: the columns returned by the query are correct, however, they are a subset of the accurate answer. Therefore possible repair rules can be defined to check the head of the query. Additionally, this may indicate the need for more expressive ontologies.

	Average Achievable Improvement
All Questions	55.57%
Low Question/Low Schema	49.30%
High Question/Low Schema	40.45%
Low Question/High Schema	72.23%
High Question/High Schema	57.70%

Table 20: Average Achievable Improvement of Overall and Quadrant Results

C.3.2 Error Type Results

In our experiments, we kept count of the number of times a rule was invoked by the OBQC. Table 21 presents the percentage of usage of each rule in the OBQC. Notably, 70% of the repairs were done by the Body rules checks while 30% of the repairs were done by the Head rules. The rules exclusively related to domain were invoked 42.16% of the time and surprisingly, rules exclusively related to range were invoked less than 1% of the time. The Domain Range rule contributed to 22.78% of the repairs.

A surprising result is that the domain related rules had the largest impact in repairs. These results may shine some light on what is happening underneath the hood inside an LLM, namely the relationship between english language and triples of a graph. English is written and read from left to right. The domain of a property has a relationship to the left side of a triple. If the LLM writes a query which is wrong, it would most probably get it wrong at the beginning of a sentence/triple. This may be an explanation on why the domain related rules were the most impactful.

Rule	**Usage**
Double Domain	37.47%
Domain Range	22.78%
IRI Output	18.40%
Subject Output	11.91%
Domain	4.69%
Incorrect Property	4.26%
Range	0.43%
Double Range	0.06%

Table 21: Rule usage in the Ontology-based Query Check

D Conclusion

We are now able to provide answers to our research questions:
RQ1: To what extent Large Language Models (LLMs) can accurately answer Enterprise Natural Language questions over Enterprise SQL databases.

Answer: Using GPT-4 and zero-shot prompting, Enterprise Natural Language questions over Enterprise SQL databases achieved 16.7% Average Overall Execution Accuracy. For Low Question/Low Schema, the Overall Execution Accuracy was 25.5%. For High Question/Low Schema, the Overall Execution Accuracy was 37.%. However, for both Low Question/High Schema and High Question/High Schema, the accuracy was 0%.

RQ2: To what extent Knowledge Graphs can improve the accuracy of Large Language Models (LLMs) to answer Enterprise Natural Language questions over Enterprise SQL databases.

Answer: Using GPT-4 and zero-shot prompting, Enterprise Natural Language question over a Knowledge Graph representation of the enterprise SQL database achieved 54.2% Average Overall Execution Accuracy. The overall SPARQL accuracy was 3x the SQL accuracy and the accuracy improvement was 37.5%.

RQ3: To what extent can the accuracy increase by leveraging the ontology of a knowledge graph to detect errors of a SPARQL query and an LLM to repair the errors?

Answer: By using the same zero-shot prompting on GPT-4 and adding the Ontology-Based Query Check and LLM Repair, that new accuracy is 72.55%. That is over a 4x accuracy improvement compared to the zero-shot promting with SQL. Given that we consider an unknown result in our approach because the LLM Repair is not able to repair and the Ontology-Based Query Check catches the error, the overall error rate is 19.44%.

RQ4: What types of errors are most commonly presented in SPARQL queries generated by an LLM?

Answer: 70% of the repairs were done by the Ontological checks while 30% of the repairs were done by the SELECT Clause checks. The domain related rules had the largest impact in repairs, being invoked 42.16% of the time. An interpretation of this result is that if the LLM writes a query which is wrong, it would most probably get it wrong at the beginning of a sentence/triple and the domain of a property has a relationship to the left side of a triple.

The answers to our research questions are evidence that supports the main conclusion of our work: Knowledge Graph provides higher accuracy for LLM powered question answering systems on SQL databases.

What is evident is that context is crucial for accuracy and these results further emphasizes the need to invest in business context. The point to be made here is a call to action that investing in context of SQL databases is required to increase the accuracy of LLMs for question answering over the SQL databases. In this work, the context was presented in the form of a knowledge graph consisting of an ontology that describes the semantics of the business domain and mappings that connect the physical schema with the ontology which are used to create the knowledge graph. The ontology can also include further semantics such as synonyms, labels in different languages, which are not expressible in a SQL DDL.

Arguably, our work has influenced the wider data industry to acknowledge the need to invest in semantics and knowledge graphs in this new AI era. Our work has been reproduced and validated by multiple independent vendors[16][17] [18] [19] [20] [21] [22], and contributed to rise of GraphRAG[23].

The takeaway for enterprises is that in order to provide trustworthy question answering systems that results in highly accurate results, they must make an investment in business context and semantics.

[16]https://roundup.getdbt.com/p/semantic-layer-as-the-data-interface

[17]https://www.atscale.com/blog/semantic-layers-make-genai-more-accurate/

[18]https://www.wisecube.ai/blog/optimizing-llm-precision-with-knowledge-graph-based-natural-language-qa-systems/

[19]https://blog.kuzudb.com/post/llms-graphs-part-1/

[20]https://delphihq.substack.com/p/delphi-at-100-dbt-semantic-layer

[21]https://cube.dev/blog/semantic-layers-the-missing-piece-for-ai-enabled-analytics

[22]https://www.stratio.com/blog/stratio-business-semantic-data-layer-delivers-99-answer-accuracy-for-llms/

[23]https://neo4j.com/blog/graphrag-manifesto/

This context needs to be effectively managed in metadata managements systems such as data catalog and governance platforms built on a knowledge graph architecture.

References

[1] Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In Proceedings of the 12th International Conference on Database Theory, ICDT '09, page 31–41, New York, NY, USA, 2009. Association for Computing Machinery.

[2] Dean Allemang, Jim Hendler, and Fabien Gandon. Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL, volume 33. Association for Computing Machinery, New York, NY, USA, 3 edition, 2020.

[3] Dean Allemang and Juan Sequeda. Increasing the accuracy of llm question-answering systems on sql databases with ontologies. In Proceedings of the International Semantic Web Conference (ISWC), 2024.

[4] Dean Allemang and Juan Sequeda. Increasing the LLM accuracy for question answering: Ontologies to the rescue! CoRR, abs/2405.11706, 2024.

[5] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. Repairagent: An autonomous, llm-based agent for program repair, 2024.

[6] Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriber. Expanding the scope of the ATIS task: The ATIS-3 corpus. Proceedings of the workshop on Human Language Technology, pages 43–48, 1994.

[7] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-SQL evaluation methodology. In Iryna Gurevych and Yusuke Miyao, editors, Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 351–360, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[8] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-sql evaluation methodology. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 351–360, July 2018.

[9] Alessandra Giordani and Alessandro Moschitti. Automatic generation and reranking of sql-derived answers to nl questions. In Proceedings of the Second International Conference on Trustworthy Eternal Systems via Evolving Software, Data and Knowledge, pages 59–76, 2012.

[10] Parke Godfrey and Jarek Gryz. Answering queries by semantic caches. In Trevor J. M. Bench-Capon, Giovanni Soda, and A Min Tjoa, editors, Database and Expert Systems Applications, 10th International Conference, DEXA '99, Florence, Italy, August 30 - September 3, 1999, Proceedings, volume 1677 of Lecture Notes in Computer Science, pages 485–498. Springer, 1999.

[11] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. Automated program repair. Commun. ACM, 62(12):56–65, nov 2019.

[12] Bert F. Green, Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: An automatic question-answerer. In Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference, IRE-AIEE-ACM '61 (Western), page 219–224, New York, NY, USA, 1961. Association for Computing Machinery.

[13] C. Cordell Green and Bertram Raphael. The use of theorem-proving techniques in question-answering systems. In Proceedings of the 1968 23rd ACM National Conference, ACM '68, page 169–181, New York, NY, USA, 1968. Association for Computing Machinery.

[14] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. ACM Trans. Database Syst., 3(2):105–147, jun 1978.

[15] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge Graphs. Synthesis Lectures on Data, Semantics, and Knowledge. Morgan & Claypool Publishers, 2021.

[16] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. ACM Comput. Surv., 54(4):71:1–71:37, 2022.

[17] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 963–973, 2017.

[18] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. Inferfix: End-to-end program repair with llms. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, page 1646–1656, New York, NY, USA, 2023. Association for Computing Machinery.

[19] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, 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), pages 2261–2273, Online, August 2021. Association for Computational Linguistics.

[20] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows, 2024.

[21] Fei Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. Proceedings of the VLDB Endowment, 8(1):73–84, September 2014.

[22] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In Alice Oh, Tristan Naumann, Amir Globerson,

Kate Saenko, Moritz Hardt, and Sergey Levine, editors, Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023.

[23] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. arXiv preprint arxiv:306.08302, 2023.

[24] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In Proceedings of the 8th International Conference on Intelligent User Interfaces, pages 149–157, 2003.

[25] Juan Sequeda, Dean Allemang, and Bryon Jacob. A benchmark to understand the role of knowledge graphs on large language model's accuracy for question answering on enterprise SQL databases. CoRR, abs/2311.07509, 2023.

[26] Juan Sequeda, Dean Allemang, and Bryon Jacob. A benchmark to understand the role of knowledge graphs on large language model's accuracy for question answering on enterprise SQL databases. In Olaf Hartig and Zoi Kaoudi, editors, Proceedings of the 7th Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA), Santiago, Chile, 14 June 2024, pages 5:1–5:12. ACM, 2024.

[27] Lappoon R. Tang and Raymond J. Mooney. Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing. In 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, pages 133–141, 2000.

[28] W. A. Woods. Transition network grammars for natural language analysis. Commun. ACM, 13(10):591–606, oct 1970.

[29] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: Query synthesis from natural language. In International Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM, pages 63:1–63:26, October 2017.

[30] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 3911–3921, 2018.

[31] John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2, pages 1050–1055, 1996.

[32] Quanjun Zhang, Chunrong Fang, Yuxiang Ma, Weisong Sun, and Zhenyu Chen. A survey of learning-based automated program repair. ACM Trans. Softw. Eng. Methodol., 33(2), dec 2023.

[33] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. CoRR, abs/1709.00103, 2017.