Increasing the Accuracy of LLM Question-Answering Systems with Ontologies

Dean Allemang(✉) and Juan Sequeda

AI Lab, data.world, Austin, USA
{dean.allemang,juan.sequeda}@data.world

Abstract. There is increasing evidence that question-answering (QA) systems with Large Language Models (LLMs), which employ a knowledge graph representation of an enterprise SQL database (Text-to-SPARQL), achieve higher accuracy compared to systems that answer questions directly on SQL databases (Text-to-SQL). The objective of this research is to further improve the accuracy of these LLM Question Answering systems. Our approach, Ontology-based Query Check (OBQC), is to check the LLM generated SPARQL query against the semantics specified by the ontology. A query will be flagged as incorrect and prevented from execution if it does not align with the ontological semantics. The study also explores the LLM's capability in repairing a SPARQL query given an explanation of the error (LLM Repair). Our methods are evaluated using the chat with the data benchmark. The primary finding is our method further increases the accuracy overall by 21.59% thus pushing the overall accuracy level to 65.63%. These results provide further evidence that investing knowledge graphs, namely the ontology, provides higher accuracy for LLM powered question answering systems. Our method is a component of the data.world AI Context Engine which is being widely used by customers in Generative AI production use cases that enable business users to chat with SQL databases.

1 Introduction

Business users and executives would like to have an expert about their business available to them at all times in order to ask questions and receive accurate, explainable and governed answers. In order to achieve this goal, there has been decades of research on question answering on structured data, namely SQL databases [1–5,7,8]. With the rise of Generative AI and Large Language Models, this desire is stronger than ever. This goal is achievable if enterprises invest in Knowledge Graphs. Our previous research provided evidence that an LLM powered question answering system that answers enterprise natural language questions over a Knowledge Graph representation of an enterprise SQL database returns 3x more accurate results than a LLM without a Knowledge Graph [9].

Enterprises will require higher accuracy in order to adopt these question answering systems. Thus there is a critical need to explore deterministic

G. Demartini et al. (Eds.): ISWC 2024, LNCS 15233, pp. 324–339, 2025.
https://doi.org/10.1007/978-3-031-77847-6_18

approaches to increase the accuracy. Without improved accuracy, we risk that question answering powered LLM systems will not receive the enterprise uptake, thus not fulfilling the original goal.

From our previous research, we found that there were some identifiable patterns to how generated queries could fail. One such pattern was that the generated query would use a property in the wrong direction; for example, an Agent is sold by a Policy (as opposed to the correct usage, a Policy is sold by an Agent). We realized that the semantics of the ontology can detect errors of this sort.

For example, given a question "return all the policies that an agent sold", resulted in a SPARQL query:

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

and given the following snippet of an ontology

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

we could determine that the generated query is correct if the domain of `:soldByAgent` is `:Policy`. However, per the query, the domain is `:Agent` and assuming they are disjoint, the generated query is incorrect.

Our approach consists of two parts. First, the **Ontology-based Query Check** (OBQC), checks query validity by applying seven heuristic rules based on RDFS semantics. These rules apply to the body of the query. Therefore the head of query (i.e. the SELECT clause) is not checked and is outside of the scope of this work. 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.

Second, the **LLM Repair**, repairs the SPARQL query generated by the LLM, by reprompting the LLM with an explanation of why the query is incorrect. The explanation comes from the rule that was fired to detect the error.

Our approach gives us the opportunity to understand the capability of an LLM to repair a SPARQL query and thus further improve the accuracy.

We evaluated our approach using the chat with the data benchmark from our previous work[1] [9]. Our primary findings are the following:

- By grouping all the questions in the benchmark, the OBQC and LLM Repair increased the accuracy by 21.59%, achieving an overall Execution Accuracy of 65.63% after repairs. This is a 10% improvement based on our previous results.
- Low complexity questions on low complexity schemas achieved 74.61% Execution Accuracy after repairs.

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

- All question types on high complex schemas, increased the accuracy between 26.9% to 43.71%, a substantial improvement.
- The rules related to domain were invoked 58% of the time. Surprisingly, rules related to range were invoked only 3% of the time.

These results further indicate the need for enterprises to invest in semantics, ontologies and knowledge graphs.
Novelty and significance of Knowledge Graphs: the use case of this work is for non-technical users to ask an LLM-powered system natural language questions of their SQL data, and get reliable answers. Our works moves the performance from unacceptable to acceptable through knowledge graph, and the formal semantics of RDFS in particular.
Uptake and Impact: The approaches presented in this paper are vital components of a new data.world product, AI Context Engine which is in production use by a variety of data.world customers.
Soundness and Quality: Our work is presented following scientific rigor: research question, hypothesis, experimental setup and evaluation that provides evidence supporting our claims.

2 Research Question and Hypothesis

The goal of our work is to understand how to increase the accuracy of Large Language Model powered systems that answer questions on a knowledge graph. Our intuition is that the ontology can be used to improve the accuracy. This intuition is based on the observation of our previous work [9]: when the LLM generated a SPARQL query that produced inaccurate results, the failure was a result of an incorrect path or direction.

Therefore, we investigate the following research question: *to what extent can the ontology of a Knowledge Graph be used to improve the accuracy of a Large Language Model (LLM) based question answering system?* This research question can be divided into two sub research questions:
RQ1: To what extent can an LLM be used to repair the errors presented in SPARQL queries generated by itself?
RQ2: What types of errors are most commonly presented in SPARQL queries generated by an LLM?

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

3 Ontology-Based Query Check

The technologies in the Semantic Web stack have been built on a rigorous logical foundation. We leverage this foundation to create an ontology-based query check system for SPARQL queries. This service takes two inputs: a SPARQL query and an ontology, and returns one output; 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 and the ability to query the ontology via SPARQL itself. If the generated query deviates from the ontology and if so, 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). 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, and OPTIONAL clauses, aren't considered since the focus is on examining the compatibility of the BGP with the ontology structure. 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, construction of a *conjunctive graph*[2], with two named graphs: `:query` and `:ontology`, representing the BGP-as-RDF and the ontology, respectively.

Third, application of ontology consistency rules guided by the formal logic of RDFS and OWL. These SPARQL-implemented rules examine the `:query` and `:ontology` graphs to identify whether the query diverges from the ontology.

3.1 Domain Rule

We will describe the OBQC process in detail with the RDFS rule for domain consistency. The domain rule is formally defined as:

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

and defined in English as: If the domain of a property p is D, then the subject of any triple using p as a predicate must be a member of D.

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 triple patterns:

```
?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:):

[2] using RDFLib nomenclature.

```
  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 following query detects a violation of the domain rule:

```
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 .
    }
}
```

Let's look at this from the point of view of RDFS.

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 binding $?s \mapsto qq : agent, ?p \mapsto : soldByAgent, ?class \mapsto : Agent$.

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 $?p \mapsto : soldByAgent, ?domain \mapsto : Policy, ?s \mapsto qq : agent, ?class \mapsto : Agent$.

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*}

3.2 Range Rule

Similar to the rdfs:domain check, we can do a check for rdfs:range.

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

The message template is: *The property* {*p*} *has range* {*range*}*, but its object* {*o*} *is a* {*class*}*, which isn't a subclass of* {*range*}

Consider the following an example: an ontology that has one more step between a Claim and a Policy than many people might expect:

```
in:against rdf:type owl:ObjectProperty ;
           rdfs:domain in:Claim ;
           rdfs:range in:PolicyCoverageDetail  .
in:hasPolicy rdf:type owl:ObjectProperty ;
             rdfs:domain in:PolicyCoverageDetail ;
             rdfs:range in:Policy  .
```

that is, claims are not made directly against a policy, instead they are made against a PolicyCoverageDetail, which has a Policy. However, the generated query missed this detail:

```
    ?policy rdf:type :Policy .
    ?claim rdf:type :Claim ;
           :against ?policy .
```

We have observed this as a typical error that an LLM makes; it is relying on its background knowledge of policies and claims, rather than on the details of the ontology. How can we tell that this is an error?

The analysis is very similar to the one for the domain rule; in this case, the range rule looks for the range of `:against`, which is `:PolicyCoverageDetail`. But in the query, we see that the object of the predicate `:against` is the variable `?policy`, which in turn, has type `:Policy`.

The template is filled out as *The property :against has range :PolicyCoverageDetail, but its object ?policy is a :Policy, which isn't a subclass of :PolicyCoverageDetail.*

3.3 Double Range Rule

What if two triples make conflicting requirements on the range of a property?

```
SELECT ?p ?rangep ?q ?rangeq WHERE {
    GRAPH <https://ai.data.world/benchmark/query#> {
       ?s1  ?p ?o .
       ?s2  ?q ?o .
    }
    GRAPH <https://ai.data.world/benchmark/ontology#>{
       ?p rdfs:range ?rangep.
       ?q rdfs:range ?rangeq.
       FILTER (ISIRI (?rangeq)) FILTER (ISIRI (?rangep))
    }

    FILTER NOT EXISTS {{
       { ?rangep rdfs:subClassOf* ?rangeq .}
       UNION
       { ?rangeq rdfs:subClassOf* ?rangep .}
}}}
```

Consider the following snippet of a generated SPARQL query

```
?claim :against ?policy .
?policy_coverage_detail :hasPolicy ?policy .
```

The LLM has tried to insert a PolicyCoverageDetail between the claim and the policy, but it didn't get it right. The range of `:against` and the range of `:hasPolicy` are `:PolicyCoverageDetail` and `:Policy` respectively. The check system outputs the following explaination: *The property :against has range :PolicyCoverageDetail, and :hasPolicy has range :Policy, and these are incompatible.*

3.4 Double Domain Rule

This is exactly like doublerange, but with domains

```
SELECT ?p ?domp ?q ?domq WHERE {
    GRAPH :query{
      ?s ?p ?o ;
         ?q ?o2 .
    }
    GRAPH :ontology{
      ?p rdfs:domain ?domp.
      ?q rdfs:domain ?domq.
      FILTER (ISIRI (?domq)) FILTER (ISIRI (?domp))
    }
    FILTER NOT EXISTS {{
      { ?domp rdfs:subClassOf* ?domq .}
      UNION
      { ?domq rdfs:subClassOf* ?domp .}
}}}
```

The template: *The property {p} has domain {domp}, and {q} has domain {domq}, and these are incompatible.*

3.5 Incorrect Property

Check to see if the generated query uses a property that is not part of the ontology.

The template is: *The property {p} isn't defined in the ontology. Please only use properties from the ontology, or from a standard source like rdf:, rdfs:, owl:, or skos:*

3.6 SELECT Clause Checks

The previous rules only check the BGP for compatibility with the ontology. 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, and can't be fixed using this method. There is one exception to this; a very common error is to include a variable in the SELECT clause that will be bound to an

IRI or a blank node. It is not usually good practice to return these as values for a query that will be viewed by a business user; there is no guarantee that it will be meaningful (and in fact, a good reason to expect that it will not be meaningful). There are some things in the ontology that can predict whether a value will be a literal; for example, the RDF spec itself insists that the subject of any triple cannot be a literal; if a query selects a variable that is the subject of a basic triple pattern, we know that is inappropriate. Similarly, if a predicate has a specified range, it is common practice for this to be a named class, whose members are named resources, i.e., IRIs. So any object of a predicate with a range can be safely inferred to not be a literal. We have extended this method to identify variables in the SELECT clause and indicate them in the conjunctive graph, so that we can detect these situations as well.

The template is: *Your selected variable {varname} is an IRI; your output should be something human readable, an ID or a label.*

Subject Output Rule

```
SELECT  ?varname WHERE {
    GRAPH :query {
      ?s ?p ?o .
      BIND  (REPLACE(STR(?s), "^.*[/#]", "") as ?varname)
      ?s a qq:Variable .   }}
```

Template: *Your selected variable ?{varname} is an IRI (the subject of a triple is always an IRI). Your output should be something human readable, an ID or a label.*

3.7 Relationship with SHACL

Our approach uses the ontology for integrity constraint validation purposes under the closed-world semantics. We could have leveraged SHACL, and to do so, we would have needed to do the following:

1. Convert a SPARQL query to RDF. This is required for our approach as well as for a SHACL approach.
2. Convert an OWL/RDFS model to SHACL. There are a number of systems to do this, but none of them are standard because this actually violates the Open World semantics of RDFS/OWL. Therefore we would have needed to evaluate different semantics. For our approach, we converted the RDFS semantics to SPARQL.
3. Select a SHACL processor. Note that usually SHACL processors will depend on a SPARQL implementation, which our approach is already leveraging.
4. Prompt Engineering: the output from the error system has to be rendered in a form consumable by an LLM. This is also required for our approach.

The steps between our approach and a SHACL approach are similar. We do acknowlege that a SHACL based approach is a viable implementation. However, we do not believe it would have contributed to different results of our work.

3.8 Limitations of Our Approach

This approach takes advantage of the power of an ontology to describe data and metadata, but it has some known limitatons.

Domains or ranges represented as union or other logical combinations of classes may break the implementation. We mentioned this above, when we noticed the filter that insisted that the domain must be an IRI. This limitation can be repaired by using more complex SPARQL queries, but trying to implement the entire logical definition of OWL in SPARQL is not only difficult, in general, it is not possible. However, for most ontologies that are actually used in industry, simple rules like the ones outlined here are sufficient.

The hierarchical checks made assume that classes not explicitly defined to be subclasses of one another are disjoint - an assumption not guaranteed by OWL's Open World Assumption. It is common practice, even in very well-governed OWL ontologies, not to include all of the disjointness axioms that are known to be true, effectively making something of a closed-world assumption in many parts of an OWL model.

4 LLM Repair: Repairing SPARQL Queries with an LLM

The output of the OBQC (the list of issues for which the query is incorrect), along with the SPARQL query, are sent to the LLM in order to repair the query. The prompt is the following:

```
We have a query {query} with some issues outlined here {issues}
Please re-write it.
```

The resulting SPARQL query 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 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 query 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.

5 Experimental Setup

The experiments reported here use the same data as the Chat with the Data benchmark from our previous work, which is available on github[3]. As a reminder,

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

that benchmark consists of 1) enterprise SQL Schema based on the OMG Property and Casualty Data Model 2) enterprise question-answer that fall on a combination of two spectrums: a) low to high question complexity pertaining to business reporting use cases to metrics and Key Performance Indicators (KPIs) questions, and b) low to high schema complexity which requires a smaller number of tables to larger number of tables to answer the question. These two spectrums form a quadrant in which questions can be classified: Low Question/Low Schema, HighQuestion/Low Schema, Low Question/High Schema, and High Question/High Schema. 3) Context Layer: An OWL ontology describing the Business Concepts, Attributes, and Relationships of the insurance domain, an R2RML mappings from the SQL schema to the OWL ontology. The ontology and mappings can be used to create a Knowledge Graph representation of the SQL database.

The question answering system we evaluated was a SPARQL zero-shot prompt to GPT-4, that is instructed to generate a query, which is executed over a virtualized knowledge graph in data.world that is mapped to a relational database. This is the same setup as in our previous work [9]:

SPARQL Zero-shot Prompt

```
Given the OWL model described in the following TTL file:
{INSERT OWL ontology}
Write a SPARQL query that answers the 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}
```

This experiment takes advantage of the query checking capabilities outlined in the previous section. An ontology can do more than advise an LLM about how to write a query; it can also evaluate whether a query "makes sense", i.e., the semantics of the query matches the ontology. We use this capablitiy in two ways: first, to find issues with a query and repair it as described above. But second, and perhaps even more importantly, this allows us to detect a query that we know for certain cannot succeed. In addition to successful queries (that get the right answer), unsuccessful queries (that get the wrong answer), we have *unknown* queries, queries that we know are incorrect. This is an important category, as it contrasts with the well-known and oft-maligned "*hallucination*" behavior of LLMs, where it provides an incorrect answer with great confidence and insistence. In this setup, we also have answers that we know to be incorrect. This third kind of result plays into all of the metrics we track in the results.

As we summarize the results, we follow the metric of Execution Accuracy (EA) from the Spider benchmark [6]. 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.

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.

6 Results

In order to provide an answer to the overall research question, *to what extent can the ontology of a Knowledge Graph be used to improve the accuracy of a Large Language Model (LLM) based question answering system?*, we present the results addressing the two specific research questions:

6.1 Results for RQ1: To What Extent Can an LLM Be Used to Repair the Errors Presented in SPARQL Queries Generated by Itself?

By grouping all the questions in the benchmark, the Execution Accuracy First Time achieves 44.04% and the Execution Accuracy with Repairs increases to 65.63%. Therefore the Ontology-based Query Check and LLM Repair increased the accuracy by 21.59%. The Execution Unknown with Repairs is 6.76% which implies that the LLM Repair is usually able to repair the queries. By combining the Execution Accuracy with Repairs and Execution Unknown with Repairs, we achieve 72.39%, which indicates that an inaccurate answer is presented 27.61%.

Note that the 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%, thus with the Ontology-based Query Check and LLM Repair, we report further increased accuracy.

We observe the following results for each quadrant:

- Low Question/Low Schema: the Ontology-based Query Check and LLM Repair increased the accuracy by 16.69%. The Execution Unknown with Repairs was the highest in this quadrant, 14.51%. Combined, this implies that an inaccurate answer is presented only 10.88% of the time.
- High Question/Low Schema: the Ontology-based Query Check and LLM Repair had a minimal impact. It's not clear why.
- Low Question/High Schema: the Ontology-based Query Check and LLM Repair had a substantial impact by increasing the accuracy by 43.71%.
- High Question/High Schema: the Ontology-based Query Check and LLM Repair had a meaningul impact by increasing the accuracy by 26.9%.

Overall, the Ontology-based Query Check and LLM Repair favorably increased the accuracy. First, Low Question/Low Schema questions can achieve 10% of inaccurate answers because 75% are accurate answers and 15% are answers that the systems is unable to answer, therefore "no answer" is better than an incorrect answer. Second, the Ontology-based Query Check and LLM Repair provides an important impact to Low and High questions on High

Schema. This is a promising result which further indicates that need for enterprised to invest in semantics, ontologies and knowledge graphs (Table 1).

These results are evidence that supports our hypothesis: *An LLM powered question answering system that answers a natural language question over a knowledge graph can use the ontology to increase the accuracy of answers.*

Table 1. Average Overall Execution Accuracy (AOEA) of Overall

	Execution Accuracy First Time	Execution Accuracy with Repairs	Execution Unknown with Repairs	Execution Accuracy + Unknown with Repairs
All Questions	44.04%	65.63%	6.76%	72.39%
Low Question / Low Schema	57.92%	74.61%	14.51%	89.12%
High Question / Low Schema	65.27%	67.71%	3.03%	70.74%
Low Question / High Schema	17.59%	61.30%	1.77%	63.07%
High Question / High Schema	29.10%	56.00%	5.77%	61.76%

A follow up question is: how much of the possible improvement did we achieve for execution accuracy? In other words, given the number of times a system achieved an accurate answer on the first time, and the total numner 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 ocurred 4 times out of the 8 possible times. Achievable Improvement is calculated as (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 2

The results indicate that the OBQC is able to succesfully repair queries close to half of the time. Thus, we are half way there and there is still room for improvement. Recall that we are checking the body of the query, therefore possible repairs may require to be in the head of the query. Therefore additional rules may need to be defined. This may indicate that more expressive ontologies can be used to fix the queries.

Table 2. Average Achievable Improvement of Overall and Quadrant Results

	Average Achievable Improvement
All Questions	45.65%
Low Question/Low Schema	42.77%
High Question/Low Schema	37.67%
Low Question/High Schema	54.62%
High Question/High Schema	46.03%

6.2 Results for RQ2: What Types of Errors Are Most Commonly Presented in SPARQL Queries Generated by an LLM?

In our experiments, we kept count of the number of times a rule was invoked by the OBQC. Table 3 presents the percentage of usage of each rule in the OBQC.

Notably, the rules related to domain were invoked 58% of the time: Double Domain rule was used 42% while Domain rule was 16%. Surprisingly, rules related to range were invoked only 3% of the time. The SELECT clause checks were invoked 34.35% of the time, which indicates that 1/3 of the repairs did not involve issues with the body of the query. This may shine some light on what is happening underneath the hood in an LLM.

Table 3. Usage of the rules in the Ontology-based Query Check

Rule	Usage
Double Domain	42.05%
IRI Output	21.76%
Domain	16.01%
Subject Output	12.59%
Incorrect Property	4.40%
Range	2.93%
Double Range	0.24%

7 Customer Adoption

It is clear that customers desire to have a *chat with your data* experience as if they are talking to an expert that understands their business and is able to provide answers that are accurate, explainable and governed. The research and results presented in this paper are based on an approach of co-innovation between the data.world AI Lab and data.world's customers, and are taking us closer to this goal. The two approaches presented in this paper, **Ontology-based Query**

Check and **LLM Repair**, are the result of a series of hackathons that the authors have conducted with various data.world customers since October 2023. The components presented in this work have been productized and are part of the data.world AI Context Engine[4], a much larger agent/planning system. The data.world AI Context Engine, an API-driven application that connects our customer's data and metadata with Generative AI to power trusted conversations with structured data, is in production use by a variety of data.world customers. All customer's questions that go through the AI Context Engine are passing through **Ontology-based Query Check** and **LLM Repair** components, thus making them vital parts of the system. The following are sample case studies:

– A travel company has public information about their cruises, ships and ports. They want to enable potential customers to ask any type of questions about this public information. For example *How many cruises are planned by a given ship in 2025 and which ports are they visiting?* and *What are all the cruises we offer in 2025, how many days does it last, when is the departure and return date, departing from which port, and on which ship is it on?*. The business value is that potential customers can learn about the offerings and customer support is also able to answer questions faster.
– A digital marketing company provides a platform for marketing analyst to understand the performance of web analytics, ad platform, and order/sales insights by asking questions such as *What percent of customers have only made a single purchase?* and *Which products do customers buy most in their first purchase? Second Purchase?*
– An adversting technology company is enabling non technical client facing users, strategists, planners to ask questions abiout audiences and brands. For example *How many households are in the market for electric vehicles, and have children, have a higher income tier grouped by the model they are in the market for?* and *What are the brand scores for Company ACME in the UK in 2022?*. The business value is that the users are able to get an answer without asking data teams, thus being able to self-service in a short amount of time.

8 Summary and Conclusion

To summarize, we first present the **Ontology-based Query Check** which checks a SPARQL query generated by an LLM against the semantics of an ontology to identify and explain erroneous queries. Then we present **LLM Repair**, which takes the explanation and the erroneous SPARQL query and prompts the LLM to repair. This cycle repeats until the check passes, or an upper limit of cycles is reached. reached. Our experimental results provide evidence that our approach increase the accuracy of Large Language Model (LLM) question-answering systems overall by by 21.59%, achieving a overall execution accuracy of 65.63%. Our findings provide evidence that the use of ontologies can further increase the accuracy of LLM question-answering systems, thus supporting

[4] https://data.world/ai/.

businesses' trust in the GenAI solutions they deploy. With these additions, we have moved toward a system that is more reliable, less prone to common errors, and ultimately more practical for usage in real-world applications. These two components presented have been productized and are part of the data.world AI Context Engine[5], which is a much larger agent/planning system and is currently being deployed at a variety of data.world customers.

Future work involves adding more rules to the Ontology-based Query Check. Current results have already increased the accuarcy to 72% [10].

These results supports the main conclusion of this research: investement in metadata, semantics, ontologies and Knowledge Graph are preconditions to achieve higher accuracy for LLM powered question answering systems on structured data.

References

1. Green, B.F., Wolf, A.K., Chomsky, C., Laughery, K.: Baseball: an automatic question-answerer. In: Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference (New York, NY, USA, 1961), IRE-AIEE-ACM '61 (Western), Association for Computing Machinery, pp. 219–224 (1962)
2. Green, C.C., Raphael, B.: The use of theorem-proving techniques in question-answering systems. In: Proceedings of the 1968 23rd ACM National Conference (New York, NY, USA, 1968), ACM '68, Association for Computing Machinery, pp. 169–181 (1968)
3. Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D., Slocum, J.: Developing a natural language interface to complex data. ACM Trans. Database Syst. **3**(2), 105–147 (1978)
4. Woods, W.A.: Transition network grammars for natural language analysis. Commun. ACM **13**(10), 591–606 (1970)
5. Dahl, D.A, et al.:Expanding the scope of the ATIS task: the ATIS-3 corpus. In: Proceedings of the Workshop on Human Language Technology, pp. 43–48 (1994)
6. Yu, T., et al.: 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, pp. 3911–3921 (2018)
7. Zelle, J.M., Mooney, R.J.: Learning to parse database queries using inductive logic programming. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence 2, pp. 1050–1055 (1996)
8. Tang, L.R., Mooney, R.J.: 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, pp. 133–141 (2000)
9. Sequeda, J., Allemang, D., Jacob, B.: A Benchmark to Understand the Role of Knowledge Graphs on Large Language Model's Accuracy for Question Answering on Enterprise SQL Databases. CoRR arxiv preprint arxiv: abs/2311.07509 (2023)
10. Allemang, D., Sequeda, J.: Increasing the LLM Accuracy for Question Answering: Ontologies to the Rescue! CoRR arxiv preprint arxiv:abs/2405.11706 (2024)

[5] https://data.world/ai/.