UniQL: Access NoSQL DB "Without a NoSQL Query"

Tianyang Liao
tl65@illinois.edu

Bohan Wu
bohanw5@illinois.edu

Feize Shi
feizes2@illinois.edu

Yifan Zhong
yzhong32@illinois.edu

Zhaoze Wang
zhaozew2@illinois.edu

ABSTRACT

As applications become increasingly complex, people have to combine different databases to accommodate various workloads. These databases usually provide diverse query interfaces which brings hardness to access such databases. In this paper, we propose a simple, and extensible solution – UniQL, which utilizes the advanced capabilities of Large Language Models (LLMs) and the Semantic Kernel[16], together with augmentations including tailored prompts and Retrieval Augmented Generation (RAG)[13]. The system not only can simplify interaction with various databases but is also easy to extend.

We developed a prototype of this solution and evaluated it using the Spider dataset, focusing on performance metrics including conversion accuracy, conversion cost, and system extensibility. In our evaluation, the baseline version of UniQL achieved an average accuracy of 55% across three different query languages. With the addition of the RAG component, UniQL's average accuracy increased to 76%. UniQL achieves the P95 latency of 15 seconds. Furthermore, UniQL demonstrated impressive extensibility with a manageable work overhead compared to other database-specific converters.

1 INTRODUCTION

Databases have been designed and optimized for various workloads, resulting in diverse implementations. They are different from index structures, execution models, and storage layouts. However, from the perspective of users, the most notable difference often lies in the query language used. Many relational database databases are compatible with SQL since it is becoming a standard. However, some databases opt for their specific query APIs due to their special data model, which significantly differs from SQL's structure and syntax.

As applications become increasingly complex, they often require simultaneously meeting diverse and sometimes conflicting data management needs. This has led to the combined use of various databases and people have to access various databases. However, differences in query languages pose significant challenges for users, who must overcome steep learning curves of query language to effectively access these varied databases. Consequently, developing a unified API or tool that facilitates seamless interaction with multiple databases would be helpful.

Numerous studies have attempted to address this challenge, yet still encounter limitations. The Extract, Transform, and Load (ETL) pipeline facilitates the formatting and integration of data from various databases into a data warehouse such as HBase[3], allowing subsequent analysis via a processing engine like Hive[21]. However, building such an ETL pipeline demands considerable engineering effort for manually mapping data across formats and leads to significant extra storage costs to maintain a duplicate of the data. In recent years, the data lake and lakehouse[4, 8], has enabled the flexible storage and analysis of both structured, semi-structured, and unstructured data. Nevertheless, these systems still suffer from issues of data governance and application integration.

An alternative approach involves keeping data stored in separate databases while providing a tool that converts SQL queries to specific query languages[1]. These tools utilize Abstract Syntax Trees (AST) for analyzing and mapping queries, which enable them to adapt different data schema. However, the analyze approach also constrains themselves to support a specific query language. Consequently, adding support for new databases requires substantial effort in coding and testing.

In this paper, we propose UniQL, a novel, simple, and extensible solution that can leverage structured frameworks like Semantic Kernel to coordinate Large Language Models (LLMs) and vector databases for converting SQL queries to various NoSQL queries. Our goal is to offer a tool that enables users to access various databases through a uniform interface, hiding the fact that these databases utilize different query APIs. Additionally, we aim to design the framework to be easily extensible, facilitating compatibility with new databases.

Contrasting to using domain knowledge for target-specific developments, UniQL utilizes the natural language processing prowess of LLMs to understand and convert SQL queries. Based on LLMs, we have developed converter plugins to structure prompts effectively to support a variety of databases and simplify extensions. In addition, pure LLMs are known to encounter some challenges such as "hallucination problems"[25]: LLMs occasionally generate content that diverges from the user input, contradicts previously generated context, or misaligns with established world knowledge. So, to enhance the LLM, we employ the Semantic Kernel framework to create an intermediary agent that interacts with LLMs on behalf of users. This agent is designed to embed the queries and extract domain knowledge and conversion examples from VectorDB, thereby enhancing the quality of the queries generated by LLMs through Retrieval Augmented Generation (RAG).

Another challenge is to verify the correctness of the result queries, specifically ensuring that these converted queries preserve the same semantics as the original SQL queries provided by users. To our knowledge, while some research has explored this issue[7, 18, 27], a robust solution for determining semantic equivalence among the queries has yet to be established. Consequently, we have designed a result-oriented testing framework that employs execution outcomes to verify the accuracy of the conversions.

With the support of our testing framework, we evaluated UniQL using the Spider dataset. The experimental results indicate that our implementation of UniQL achieves a conversion accuracy of approximately 70% on MongoDB and Neo4j, with a P95 latency of

15 seconds. Extending our system to support additional databases is straightforward; creating a basic plugin requires only about 10 lines of code for a prompt template, and the number of knowledge sources and examples is tunable. Additionally, we investigate the characteristics of SQL queries and explore the factors that influence the conversion capabilities of UniQL.

Contributions. The contributions of this paper are as follows. We:

- Introduces a novel, simple, and extensible solution leveraging Semantic Kernel to utilize Large Language Models (LLMs) and vector databases, enabling the conversion of SQL queries into various NoSQL queries. (§3)
- Design and implement a result-oriented evaluation framework that utilizes the execution result to verify the semantic equivalence between converted results and user-provided SQL queries. (§4.1)
- Evaluate the performance of query conversion with UniQL, demonstrating the feasibility of using LLMs to convert a SQL query into another database-specific query. (§4)
- Investigate the factors for limiting conversion accuracy, which can potentially help with further improvement on the conversion accuracy. (§5)

2 RELATED WORK

Here, we first discuss some related works that can help to access heterogeneous databases and analyze their limitations. Then we provide some background knowledge about the techniques used to build UniQL.

2.1 ETL Pipelines and Data Warehouse

The Extract, Transform, and Load (ETL) pipeline plays a crucial role in performing analysis of heterogeneous databases [9, 23]. In short, data from various databases in different formats can be aggregated into a 'wide table' in a data warehouse through the ETL pipeline.

Typically, the workflow proceeds as follows: data is first extracted from various databases, then loaded into a stream processing engine where it undergoes cleansing and is transformed into a uniform format. Ultimately, this processed data is inserted into a data warehouse, such as HBase[3]. Subsequently, some warehouses like Hive[21], can even help people to use an SQL-like query language to perform further analysis.

Although we can access data from multiple databases by combining ETL and data warehouse, it suffers from some limitations. Firstly, data warehouses typically require predefined data schemas (schema-on-write), and any modifications necessitate manual changes to the code of the pipeline. This restricts the flexibility of this approach. Additionally, data warehouses introduce additional storage overhead because the data warehouse is a copy of data.

Moreover, data updates are typically periodic, and may not necessarily reflect the real-time status of the data. Thus, we still need a solution to access databases with their native APIs.

2.2 Data Lake and Lakehouse

A data lake is a centralized repository that allows users to store all their structured, semi-structured, and unstructured data. It enables the storage of raw data in its native format and uses metadata to reconstruct "schema" at runtime (schema-on-read). Data lakes usually store data on object storage like Amazon S3[2], Azure Data Lake Storage[15], and Google Cloud Storage[10]. However, data lake also have some drawbacks, such as the potential for turning into "data swamps" if not properly managed.

Data Lakehouse is an architecture that combines the flexibility of data lakes with the management features of traditional data warehouses. File formats such as Parquet[5] introduce a tighter schema for data lake tables and a columnar format to improve query efficiency, while technologies such as Delta Lake[8] and Apache Hudi[4] bring greater reliability to write/read transactions in data lakes. This drives them closer to the ACID characteristics emphasized by traditional database technologies. The advantages of data lakehouse systems include improved data governance and the ability to handle diverse data types efficiently.

2.3 Query Translation/Conversion

In addition to storing data together to ease access, an alternative approach is query translation/conversion[14]. This means that data is still stored in separate databases, and queries written in a user-preferred query language are converted into the format required by the database's query API.

Most studies/tools[1, 17, 20, 26] utilize either an Abstract Syntax Tree (AST) or a Parse Tree (PT) in their methodologies. Typically, these works employ lexical and syntax analyzers to parse SQL queries into an AST. Subsequently, the AST is used to break the SQL query into its constituent clauses. Utilizing data metadata and some rules, they map each SQL component to its NoSQL counterpart. Finally, a generator is employed to produce the corresponding NoSQL query.

Compared to a centralized "wide table," query conversion reduces computation and storage overhead but also faces certain limitations. First, the semantics and features supported by different databases can vary significantly, which is reflected in their respective query languages/APIs. Users should be made aware of these differences. For instance, some NoSQL databases do not natively support joins. Additionally, this approach is database-specific and challenging to adapt for compatibility with other databases. UniQL is inspired from this kind of tools and goes one step further. Our solution aims to address this extensibility issue, making it easier to extend support to new databases.

2.4 Large Language Model

Large Language Models (LLMs) like OpenAI's GPT [6] and Meta AI's LLaMA [22] are now popular in tasks related to syntax and semantics. GPT models are known for their ability to generate contextually relevant text by learning from diverse data inputs , while LLaMA offers a scalable architecture that efficiently handles various natural language processing tasks. Also, before those works, the versatility of large models in linguistic tasks is supported by additional research indicating that LLMs can successfully convert all language problems into a unified text-to-text format[19].

Given that conversion tasks also need to mine semantics within query language, we believe LLM will be qualified as a fundamental compute engine.

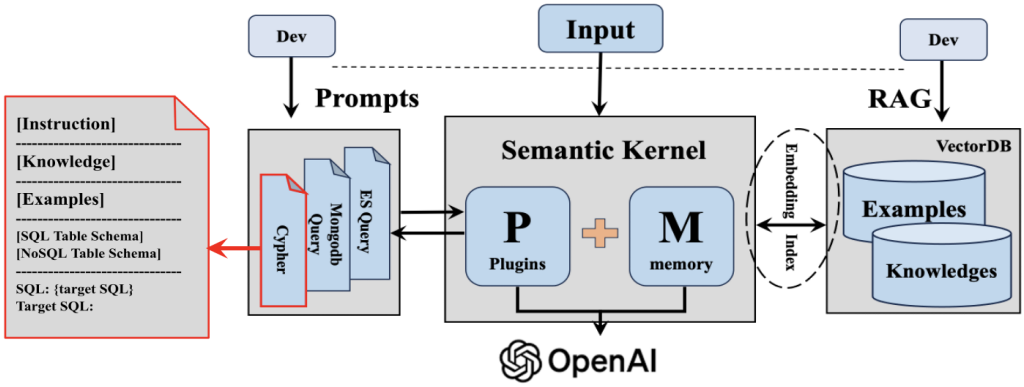

Figure 1 A schematic of the UniQL converter showcasing the Semantic Kernel at its heart, which utilizes plugins and memory to translate SQL queries into NoSQL counterparts, enhanced by VectorDB's RAG for context-rich query conversions.

3 SYSTEM DESIGN

UniQL is an extensible query converter. In this section, we will first introduce the architecture and workflow of our converter. Then, we will explain each component in detail.

3.1 Overview

As Figure 1 shows, our framework is based on large language models, enhanced with an agent framework and prompt engineering technologies such as RAG.

Our system is designed to provide a unified interface that allows users to seamlessly convert SQL queries into various NoSQL queries. This design ensures that adapting to new query languages can be achieved rapidly with minimal or even no code-level development. Traditional translation solutions, such as those utilizing an Abstract Syntax Tree (AST) or Parse Tree (PT), are typically not feasible as they are ad-hoc and tailored to specific scenarios. Instead, UniQL employs AI models, specifically large language models (LLMs), as its core computational engine. These models are extensively pre-trained and possess robust generative capabilities.

Upon the conversion model, UniQL has a fine-tuned agent that converts the LLM from a general-purpose helper to a domain-specific expert. After LLMs became popular, some excellent agent frameworks emerged, including Langchain[12] and Semantic Kernel. UniQL's agent is derived from Semantic Kernel.

This agent integrates widely-used features into our system kernel. It specifies how users interact with the system and then how the system interacts with the LLM. It also implements the two most important prompt engineering technologies, namely plugin and memory. The former allows UniQL to adapt to new query languages without development - only a new plugin needs to be uploaded. The latter facilitates generation and conversion by indexing and providing related domain knowledge.

3.2 Plugins

We utilize plugins to enhance our system's ability to convert SQL queries into various NoSQL formats. Each plugin is tailored for the conversion tasks associated with a specific type of NoSQL query. To integrate a new query language, developers can simply add new plugins without requiring modifications to the existing code. Furthermore, these plugins can be deployed dynamically, even while the system is operational, enhancing the system's flexibility and user convenience.

To implement these plugin templates, UniQL leverages the native plugin module of Semantic Kernel. Each plugin comprises two key components: a pure text file, which serves as the template for the prompts, and a JSON file, which functions as the configuration file. The configuration files manage the fields that populate the prompts and define the parameters of the LLM, such as temperature. The template files typically consist of five parts, as illustrated on the left-hand side of Figure 1.

- Explanation of tasks. This section specifies the purpose of the plugin. typically in a straightforward format, such as "Please convert the given SQL query to a MongoDB query".
- Retrieved domain knowledge to assist with conversion. This is the first part of the RAG content, consisting of snippets of official from official documents, blogs, and troubleshooting guides.
- Retrieved examples to aid in conversion. This is the second part of the RAG content, comprising manually converted examples where each example includes a SQL query and its corresponding NoSQL-version query.
- Table schema. This section provides schema information for relevant tables in both the original SQL database and the target NoSQL database.
- Final question. In this part, the prompt presents the original SQL query and the entry for the desired converted query. The LLM is expected to respond at this point, in completion generation mode.

3.3 Memory

UniQL also applies RAG to enhance accuracy, as shown in Figure 2. For each type of NoSQL query, there are useful official documents snippets that guide the conversion process, specifying the syntax and semantics. Such guidance is especially important when the target query language is not widely used and the LLM lacks a sufficient corpus to achieve high accuracy.

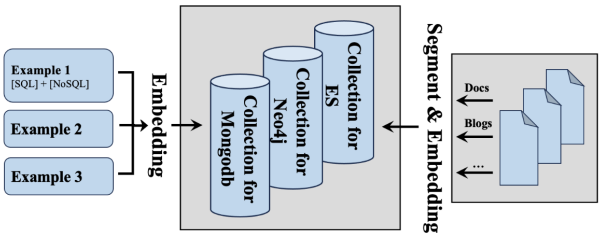

Figure 2 An outline of the RAG Structure, mapping the progression from SQL-to-NoSQL conversion examples to their embeddings, and interfacing with MongoDB and Elasticsearch collections for enhanced document retrieval.

To implement such RAG functionalities, UniQL utilizes the native memory module of Semantic Kernel. This module provides both embed and index APIs, which allow for 'semantic' searches that compare meanings directly with the given query. To store the embedding results, which represent text information as long vectors of numbers, UniQL uses a vector database. This vector database is expected to accelerate the indexing and search processes.

There are two types of RAG content for memory: examples and knowledge. Knowledge consists of official documents, blogs, and troubleshooting guides. To avoid excessively long entries, which may lead to token overflow and distraction, they are truncated into pieces. Examples include complex queries and their corresponding converted versions. To collect such examples, we applied a training dataset to our converter system during the early development stage and selected the incorrect cases. For each conversion, some memories are retrieved and used to fill the prompt template."

4 EVALUATION

In this section, using the Spider dataset we evaluate UniQL in different dimensions. Specifically, we aim to answer the following questions:

- How does UniQL with RAG outperform basic LLMs? (§4.5)
- How does UniQL perform under different kinds of queries? (§4.6)
- How easy is it to adapt UniQL for various databases?(§4.7)
- How much time does UniQL need to convert a query?(§4.8)
- What is the trade-off between cost and accuracy when applying different models?(§4.9)

4.1 Evaluation Framework

Spider's datasets focus on conversion tasks between natural language and SQL and thus do not provide equivalent NoSQL queries. Therefore, to evaluate the conversion accuracy, one challenge is how to verify the correctness of the result queries returned by the converter, or more formally, assessing the semantic equivalence between a result query and an original SQL query. Although some studies have investigated related problem[7, 18, 27], a robust and practical method for assessing the semantic equivalence of queries, especially across different query languages, has yet to be developed.

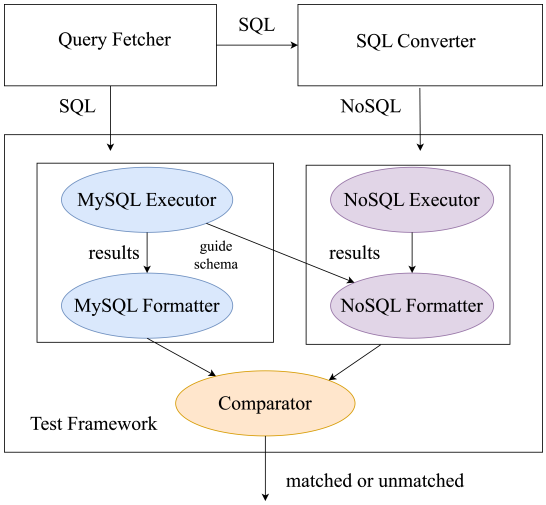

Figure 3 The architecture of evaluation framework. This framework evaluates the conversion with modules including executor, formatter and comparator.

Consequently, we aim to develop a results-driven testing framework that leverages execution outcomes to confirm the correctness of these conversions.

Statistically, if two queries return exactly matched results, there will be a high probability that they are also semantically equivalent. This conclusion becomes even more convincing when the queries are tested against large-scale data that exhibits significant variation.

Additionally, the framework should be able to automatically run all test cases and provide results, minimizing manual intervention as much as possible.

Therefore, we designed and developed a unified evaluation framework for all kinds of target databases and queries. It helps with the automatic and large-scale evaluation. It is designed for the following considerations.

As depicted in Figure 3, the evaluation framework workflow is as follows. The process starts with the QueryFetcher, which retrieves original SQL queries from the pre-processed Spider dataset, in batch and automatically. These queries are subsequently passed to the SQLConverter, where SQL queries are converted into target NoSQL queries. Then the original SQL queries and the converted queries will be executed in the MySQL engine and a target database engine respectively. Comparing the execution results from MySQL and NoSQL databases is not intuitive since NoSQL databases have heterogeneous data models. The storage may be document-based, column-based, or graph-based. The result format also depends on the implementaion of query engine. Therefore, results from both executions are then formatted by their respective formatters into a common comparable format (e.g., record-based JSON), allowing the comparator to assert whether the results match efficiently. Also, there's one more thing worth noticing, after formatting, comparator will hash each record as a whole to easy the match process. This framework is crucial for verifying that translated queries produce the same results as the original queries. It demonstrates our translations achieve semantic equivalence.

4.2 Evaluation Setup

The UniQL converter and evaluation framework are built with Python. The converter is based on semantic kernel, which acts as an agent for the LLMs offered by OpenAI. For the RAG implementation, it utilizes a Chroma vector database to store embeddings that represent specific domain knowledge for each type of conversion tasks. The evaluation framework assesses the semantic equivalence between two queries from different query languages by comparing the results' hashes. We will discuss this in detail in Section 4.4.

The specific versions of the databases used in the setup are listed, including MySQL (version 8.0.36), MongoDB (version 7.0.7), Elasticsearch (version 8.13.0), Neo4j (version 5.18.1), and Chroma (version 0.4.23).

Furthermore, the experiments were conducted on the Google Cloud Platform (GCP), utilizing an e2-standard-2 type compute engine instance. This instance is equipped with 2 vCPUs and 20 GiB of memory, providing a standardized environment for testing and evaluation.

Unless otherwise specified, all experiments in this section are based on the API version of GPT-4-turbo.

4.3 Evaluation Metrics

To evaluate the performance of our converter, we have selected some metrics to evaluate performance in the following 3 dimensions: accuracy, cost, and system extensibility.

4.3.1 Conversion Accuracy. The accuracy of the translations acts as a metric for evaluating the quality of the NoSQL queries generated by UniQL's converter. Given that Spider lacks the corresponding NoSQL queries we require, we have designed a framework (to be elaborated later) that determines whether an SQL query and a NoSQL query are semantically equivalent by comparing their execution results. The calculation of accuracy is based on the ratio of accurately translated NoSQL queries to the overall quantity of queries converted.

4.3.2 Conversion Cost. Another important perspective to consider when we evaluate the converter performance is the cost. Instead of simply focusing on optimizing the accuracy, it is of practical value to consider how much cost the high accuracy will possibly bring. The cost in this work is classified as temporal cost and financial cost.

Temporal Cost: end-to-end latency of converting a query
Financial Cost: the price of utilizing the OpenAI API

4.3.3 System Extensibility. The extensibility serves as a metric to tell us how easily UniQL can be adapted to support translation for new databases. Like many systems, we use lines of code (LOC) to approximately describe the development required to integrate support for additional databases, measuring the complexity and scalability of adaptations of our LLM-based converter.

4.4 Datasets

Spider[24] is a cross-domain semantic parsing and text-to-SQL dataset created by Yale University. It comprises 10,181 questions and 5,693 unique SQL queries spanning 200 databases across 138 domains. As illustrated in Figure 4, only 43% of the SQL queries from the Spider dataset are simple, containing only basic select and where clauses. The remaining queries incorporate more complex SQL features such as joins, aggregations, subqueries, or various combinations of these features. In our experiments, we utilized Spider's SQL queries as input and its SQLite files to construct our databases. Specifically, we employed over 50 SQLite databases and more than 1250 queries from the Spider dataset.

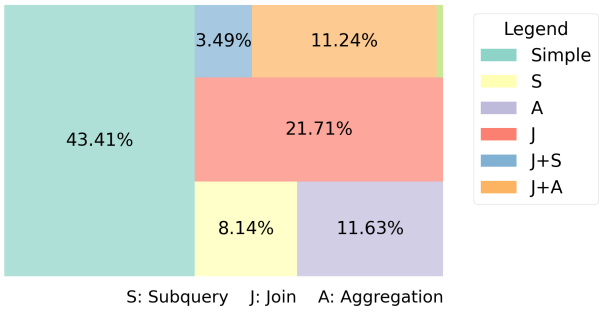

Figure 4 A treemap representation of the proportions of various types of SQL queries we use in evaluation. It illustrates the complexity of SQL queries employed: most queries contain joins, aggregations, subqueries, or arbitrary combinations of these features.

As mentioned before, we assess the semantic equivalence between an original SQL query and its converted results by comparing their execution outcomes. Thus, it is necessary to create target databases with related data.

Starting with the SQLite database files provided by the Spider dataset, our first step in the experimental setup involves migrating the data from these files to MySQL. To facilitate this conversion, we utilize the sqlite3-to-mysql tool[11], which ensures the integrity of the database schemas and data throughout the migration process.

Once the MySQL databases are established, we proceed with the migration of this data to our selected NoSQL databases: MongoDB, Elasticsearch, and Neo4j, using a combination of migration tools and custom scripts. One of the most significant challenges in this migration is adapting the relational schema from MySQL to the schema-less or flexible schema models used by NoSQL databases.

4.5 How does UniQL with RAG outperform basic LLMs?

In this section, we aim to evaluate how UniQL with RAG can enhance conversion accuracy compared to the baseline (AKA. pure prompt). The baseline configuration bypasses the RAG components of UniQL, resulting in only conversion instructions in prompt being sent to the GPT-4 API without the inclusion of domain knowledge and examples. In contrast, UniQL with RAG incorporates domain knowledge, which helps refine GPT's inference. Both approaches are evaluated across three target databases: MongoDB, Neo4j (Cypher), and Elasticsearch.

As depicted in Figure 5, the baseline converter, which employs a straightforward prompt, achieves an accuracy of 57% for MongoDB

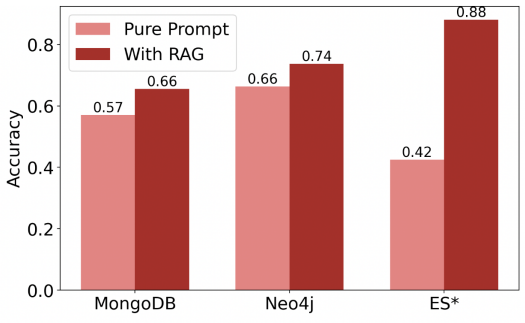

Figure 5 The bar graph compares query conversion accuracy between pure prompt and RAG-augmented methods for MongoDB, Neo4j, and Elasticsearch, highlighting significant accuracy gains with RAG, especially for Elasticsearch.

and 66% for Neo4j. With the incorporation of RAG, conversion accuracy is further enhanced to 66% for MongoDB and 74% for Neo4j. It is important to note that the accuracy figures for Elasticsearch are calculated by excluding all queries that contain 'JOIN' clauses, as Elasticsearch does not support operations equivalent to 'JOIN' very effectively, necessitating modifications to the schema of indexes or the creation of custom scripts.

We observe that UniQL with RAG significantly improves the conversion accuracy for Elasticsearch from 42% to 88%. This enhancement relies on the inclusion of schema information, which helps correct field name errors in queries. More importantly, we provide valuable knowledge from vector databases that LLMs might overlook. For instance, it is essential to set the size field to 0 in an aggregation query to retrieve only the aggregation results, and the 'keyword' should be added in the search field.

4.6 How does UniQL perform under different kinds of queries?

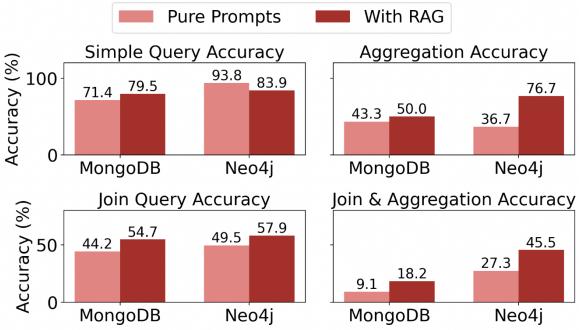

Figure 6 Bar graph showing the enhanced accuracy of RAG-augmented over pure prompt translations across simple, join, and aggregation queries in MongoDB and Neo4j.

* For Elasticsearch indicates that accuracy figures exclude queries with 'JOIN' clauses, as Elasticsearch does not effectively support these operations.

In the previous section, we provided an overview of the conversion accuracy. Now, we aim to further investigate how the conversion accuracy of UniQL varies across different types of queries. We classify the queries into the following classes (may overlap):

- **Simple Queries:** This class includes all queries of basic SQL statements with only SELECT and WHERE clauses, which exclude the 'JOIN' keyword and aggregate function.
- **Join Queries:** This class encompasses queries that contain the keyword 'JOIN' but do not include aggregate functions in their original SQL format. The purpose of these test cases is to evaluate UniQL's capability to understand the relationships between different entities.
- **Aggregate Queries:** This class consists of queries that exclusively contain aggregate functions and do not include the keyword 'JOIN' in their original SQL formulation. The objective of these test cases is to assess whether the target NoSQL query accurately preserves the semantics of SQL aggregate functions.
- **Join & Aggregate Queries:** This class comprises queries that feature both the 'JOIN' keyword and aggregate functions in their original SQL query. These test cases are designed to assess whether UniQL can accurately convert complex and comprehensive queries.

Based on the classes mentioned earlier, we evaluate the conversion accuracy for MongoDB and Neo4j, as Elasticsearch does not adequately support the join clause. The results displayed in Figure 6 depict the variance in accuracy when converting SQL queries into different NoSQL query languages. Simple queries achieve the highest accuracy, reaching up to 93.8% for Neo4j. However, for more complex queries involving both join and aggregate operations, the accuracy significantly drops. Accuracy going as low as 9.1% for MongoDB when converting queries that include both 'JOIN' and aggregation, using UniQL without RAG. This outcome underscores the challenges posed by these complex conversions.

The introduction of the RAG component significantly improves the conversion accuracy, especially for Neo4j, where it effectively handles JOIN operations. For MongoDB, while RAG also enhances accuracy, its impact is less pronounced than in Neo4j, indicating a variation in how each database system benefits from the implementation of RAG.

4.7 How easy to extend the UniQL to support different databases?

In this section, we will illustrate how effortlessly our converter can be extended to accommodate new databases as targets for conversion.

The first column of the Table 1 shows the minimal number of lines of code required for each database plugin. In UniQL, we have a template for prompts, which means each plugin can only include some specific instructions for the prompt. As a result, approximately 10 lines of instruction are sufficient to develop a plugin. The number of examples and the extent of knowledge incorporated are adjustable and may vary across different databases. For our implementation, we have provided each database with 15 examples to ensure a consistent basis for comparison.

	Lines of Codes	Amount of Examples	Document
MongoDB	8	15	...
Neo4j	7	15	...
Elasticsearch	10	15	...

Table 1: A comparative summary showcasing the work needed to develop plugins for MongoDB, Neo4j, and Elasticsearch.

In summary, the table highlights the relatively low complexity involved in developing plugins for these databases, making UniQL an extensible design.

4.8 How much time does UniQL take to convert a query?

To be more practical, a system must consider the execution time; therefore, we evaluate the conversion time of queries. Figure 7 presents the Cumulative Distribution Function (CDF) graph of conversion times for thousands of queries using the GPT-4 API. The results indicate that the 99th percentile (P99) is approximately 20 seconds, and the P95 is about 15 seconds, suggesting that UniQL is efficient enough for production use.

Now we reveal several valuable insights. First, there exists a slight difference among conversion tasks for various databases. Neo4j exhibited the fastest conversion times, followed by MongoDB, and then Elasticsearch. This variation is attributed to the different average sizes of queries required for identical tasks across these databases. Typically, Elasticsearch queries are longer, while Cypher queries for Neo4j are shorter. As a result, the time taken by the LLM API to generate responses varies, since producing longer text involves more complex processing to maintain coherence and relevance in the output. Secondly, when analyzing the conversion time for queries for the same database, it becomes evident that the time varies among different types of queries. For instance, queries containing a "JOIN" clause typically require a longer conversion time compared to simpler queries.

This experiment helps us understand the factors that influence conversion time.

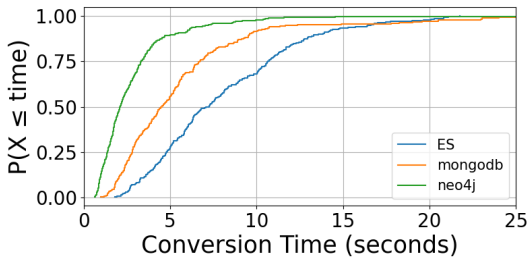

Figure 7 CDF of Conversion Time for Queries of Different Target Databases

4.9 What is the trade-off between cost and accuracy when applying different models?

Identifying the trade-off between costs and conversion accuracy can provide helpful insight for user to choose different system settings. In this experiment, costs are classified as temporal cost (time) and financial cost (price).

We construct four batches of SQL queries that differ in complexity. We use both the GPT-3.5 model and the GPT-4 model to converter these query batches. As Figure 8 and Figure 9 illustrate, the converter that uses the GPT-4 model tends to push the Pareto frontier in a direction where both accuracy and cost are increasing. This suggests that although the converter that uses GPT-4 model achieves 1.4x accuracy, it generally consumes 2.9x time and 9.9x price, compared to the converter that uses GPT-3.5. These experiments indicate a substantial cost differential to achieve better accuracy, suggesting its potential utility in high-stakes scenarios where accuracy is critically valued over cost constraints.

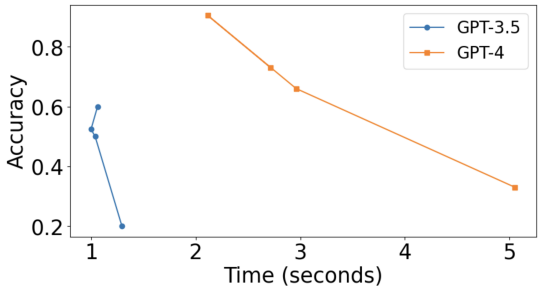

Figure 8 Conversion Accuracy and Time Trade-off for converter based on GPT-3.5 and GPT-4

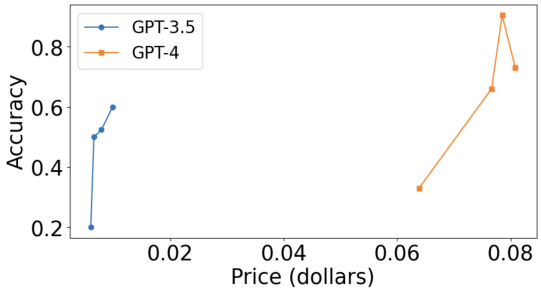

Figure 9 Conversion Accuracy and Price Trade-off for converter based on GPT-3.5 and GPT-4

5 DISCUSSION

Although the evaluations have demonstrated UniQL's capability to handle query conversion tasks effectively, which is explained by the power of LLMs and the incorporation of domain knowledge, there are still some limitations and areas for future work that need to be discussed.

5.1 Limitations

The effectiveness of UniQL in converting SQL queries to NoSQL queries for MongoDB and Neo4j with an approximate accuracy of as high as 70%. However, the 30% of cases where conversions fail highlight areas where our system needs further improvement. One critical factor contributing to these failures is the retrieval-augmented generation (RAG) approach, which uses previously compiled domain-specific examples to generate responses. While this method enhances response quality by providing contextually relevant information, its reliance on the existing examples and knowledge base limits its effectiveness. The performance is largely dependent on the quality of the provided examples and knowledge. Inappropriate selections can reduce effectiveness, especially for complex queries. To further improve conversion accuracy, work should focus on more effective RAG content and seek better indexing/similarity algorithms for the vector database.

Another notable observation is that accuracy varies significantly among target databases. For instance, databases like MongoDB are more widely used, which means that the LLM is more effectively pre-trained on related knowledge, naturally leading to higher accuracy in MongoDB queries. Additionally, some query languages like Cypher are more similar to SQL, which also facilitates the conversion process. However, for some relatively niche databases, LLMs lack knowledge about according query language, which leads to a greater reliance on RAG, further exacerbating the issues mentioned before.

Moreover, some features are not supported by certain query languages. For example, Elasticsearch does not support operations equivalent to 'JOIN.' Consequently, conversions involving such queries are likely to fail.

5.2 Future works

Looking forward, several potential aspects exist for the further development of UniQL.

- **Incorporate additional database operations:** Extend UniQL to support a broader range of database operations, such as insertions, updates, and deletions. This is crucial for complete database management and interaction.
- **Support for more NoSQL databases:** Expand the range of supported NoSQL databases to include more variants such as key-value stores, wide-column stores, and more specialized graph databases. This expansion would cater to a broader set of use cases and enhance the system's versatility.
- **Evaluate with other LLM models:** Evaluate the performance of other large language models like Llama 3, Claude 3, and Gemini. This could provide insights into their effectiveness in query translation tasks. Comparing these models may reveal strengths and weaknesses that could guide further optimization of the translation mechanisms.
- **Refactor Plug-in Architecture:** Further simplify the plug-in architecture to facilitate easier integration of additional databases, thereby extending the system's applicability and ease of use.

6 CONCLUSION

In this paper, we introduced UniQL, a query conversion system utilizing LLMs and other prompt technologies. UniQL effectively bridges SQL and various NoSQL databases, facilitating seamless query conversion that enhance interoperability and simplify data operations. Our comprehensive evaluations confirm UniQL's high accuracy and extensibility in handling complex conversions, demonstrating its ability to maintain semantic integrity across different database systems.

The integration of plugins and RAG significantly enhances knowledge retrieval, improving the precision of query conversions and enabling UniQL to adapt quickly to new NoSQL environments. This positions UniQL as a key tool in advancing database management technologies, particularly as databases continue to evolve and diversify.

REFERENCES

[1] 3T Software Labs. 2023. Studio 3T for MongoDB. Online. Available at https://studio3t.com/.

[2] Amazon Web Services. 2023. Amazon Simple Storage Service (S3). Online. Available at https://aws.amazon.com/s3/.

[3] Apache Software Foundation. 2023. Apache HBase. Online. Available: https://hbase.apache.org/ [Accessed: Insert the date you accessed the site here].

[4] Apache Software Foundation. 2023. Apache Hudi. Online. Available at https://hudi.apache.org/.

[5] Apache Software Foundation. 2023. Apache Parquet. Online. Available at https://parquet.apache.org/.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[7] Shumo Chu, Brendan Murphy, Jared Roesch, Alvin Cheung, and Dan Suciu. 2018. Axiomatic foundations and algorithms for deciding semantic equivalences of SQL queries. *arXiv preprint arXiv:1802.02229* (2018).

[8] Databricks. 2023. Delta Lake. Online. Available at https://delta.io/.

[9] Shaker H Ali El-Sappagh, Abdeltawab M Ahmed Hendawi, and Ali Hamed El Bastawissy. 2011. A proposed model for data warehouse ETL processes. *Journal of King Saud University-Computer and Information Sciences* 23, 2 (2011), 91–104.

[10] Google Cloud. 2023. Google Cloud Storage. Online. Available at https://cloud.google.com/storage.

[11] Tine Kos. 2023. sqlite3-to-mysql. https://github.com/techouse/sqlite3-to-mysql. Accessed: yyyy-mm-dd.

[12] LangChain. 2023. LangChain: Open-Source Tools for Language AI. https://www.langchain.com/.

[13] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 793, 16 pages.

[14] Mohamed Nadjib Mami, Damien Graux, Harsh Thakkar, Simon Scerri, Sören Auer, and Jens Lehmann. 2019. The query translation landscape: a survey. *arXiv preprint arXiv:1910.03118* (2019).

[15] Microsoft. 2023. Azure Data Lake Storage. Online. Available at https://azure.microsoft.com/en-us/services/storage/data-lake-storage/.

[16] Microsoft. 2024. Semantic Kernel. https://github.com/microsoft/semantic-kernel GitHub repository.

[17] B Namdeo and U Suman. 2022. A Middleware Model for SQL to NoSQL Query Translation. *Indian Journal of Science and Technology* 15, 16 (2022), 718–728.

[18] M. Negri, G. Pelagatti, and L. Sbattella. 1991. Formal semantics of SQL queries. *ACM Trans. Database Syst.* 16, 3 (sep 1991), 513–534. https://doi.org/10.1145/111197.111212

[19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[20] Geomar A. Schreiner, Denio Duarte, and Ronaldo dos Santos Mello. 2020. Bringing SQL databases to key-based NoSQL databases: a canonical approach. *Computing* 102, 1 (jan 2020), 221–246. https://doi.org/10.1007/s00607-019-00736-1

[21] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. 2009. Hive: a warehousing solution over a map-reduce framework. *Proc. VLDB Endow.* 2, 2 (aug 2009), 1626–1629. https://doi.org/10.14778/1687553.1687609

[22] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[23] Panos Vassiliadis, Alkis Simitsis, and Spiros Skiadopoulos. 2002. Conceptual modeling for ETL processes *(DOLAP '02)*. Association for Computing Machinery, New York, NY, USA, 14–21. https://doi.org/10.1145/583890.583893

[24] Tao Yu et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. https://github.com/taoyds/spider. Github.

[25] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren's song in the AI ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219* (2023).

[26] Gansen Zhao, Qiaoying Lin, Libo Li, and Zijing Li. 2014. Schema conversion model of SQL database to NoSQL. In *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. IEEE, 355–362.

[27] Qi Zhou, Joy Arulraj, Shamkant Navathe, William Harris, and Dong Xu. 2019. Automated verification of query equivalence using satisfiability modulo theories. *Proc. VLDB Endow.* 12, 11 (jul 2019), 1276–1288. https://doi.org/10.14778/3342263.3342267

A APPENDIX

A.1 Change log

- Abstract
  Add reference. Refine the description of system overview. Add a summary of our evaluation result here.
- Introduction
  We reconstruct the introduction to include an overview of the background, related work and bit flips, system overview, challenge, and evaluation takeaways.
- Related Work
  We stand at a higher view and describe related work that can help to provide a uniform interface to access different databases.
- System Design
  Rephrase our introduction of the system. Revise Figures.
- Evaluation
  Update the results and add more metrics. Add analysis of results and insight we gain.
- Discussion
  The whole section is complete at this stage.
- Conclusion
  The whole section is complete at this stage.
- Overall
  We refine most descriptions and remove verbose or ambiguous content. Most content in the draft paper is rewritten.

A.2 Github repository

https://github.com/vinland-avalon/SQLLMConverter