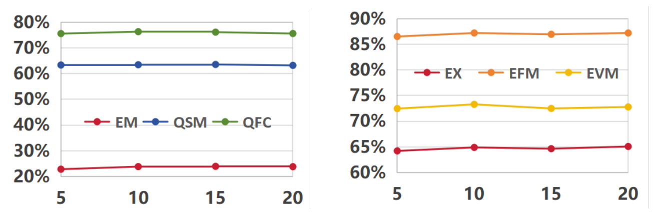
Technical Report of Paper “Bridging the Gap: Enabling Natural Language Queries for NoSQL Databases through Text-to-NoSQL Translation”

Jinwei Lu*, Yuanfeng Song‡, Zhiqian Qin*, Haodi Zhang‡§ Chen Jason Zhang*, Raymond Chi-Wing Wong¶
*The Hong Kong Polytechnic University, Hong Kong, China †WeBank Co., Ltd, Shenzhen, China
‡Shenzhen University, Shenzhen, China §HKUST(GZ), Guangzhou, China ¶HKUST, Hong Kong, China

I. Parameter Study

To explore the performance of SMART under different parameters, we conducted parameter experiments on the test set of the TEND dataset, varying the number of retrieval examples. As shown in Figure 1, we found that as the number of retrieval examples increased, SMART exhibited different performance curves across various metrics. Specifically, the accuracy of SMART on query-based metrics initially decreased and then increased. In contrast, the execution-based metrics showed a fluctuating trend. Overall, EX is the metric that best reflects the model's performance in real-world scenarios. When focusing on the EX metric, SMART achieved the highest execution accuracy of 65.08% with 20 retrieval examples.



(a) Query-based Metric (b) Execution-based Metric

Fig. 1: Parameter study. These are the variation curves of SMART under different numbers of retrieval examples and different metrics. The vertical axis represents the model's accuracy under a specific metric, and the horizontal axis represents the number of retrieval examples.

II. Analysis of SQL-to-NoSQL Approaches

Text-to-SQL, a classic task in the field of natural language processing, aims to automatically convert natural language queries into structured query language (SQL). It has a long research history and has accumulated a wealth of achievements [Yu et al.(2018)], [Finegan-Dollak et al.(2018)], [Pourreza and Rafiei(2023)], [Wang et al.(2020)]. However, with the widespread adoption of NoSQL databases in various application scenarios, exploring how to extend Text-to-SQL technology to the NoSQL domain through a cascading approach is of significant importance. Specifically, this process can be divided into multiple stages: First, existing Text-to-SQL models (such as SQLNet, TypeSQL, RAT-SQL, etc.) are used to convert natural language queries into SQL queries. For example, given the natural language query ”*Find all users older than 30*”, the model generates the corresponding SQL query: “`SELECT * FROM users WHERE age > 30`”. Subsequently, based on the syntactic mapping relationship between SQL and NoSQL, the generated SQL query is converted into the query language of the target NoSQL database. This process requires in-depth analysis of the syntax and semantics of both query languages. For instance, for the document-oriented database MongoDB, the SQL “SELECT” statement can be mapped to the “find” operation, and the ‘WHERE‘ condition can be mapped to a query filter, thereby transforming the aforementioned SQL query into a MongoDB query: “`db.users.find({age:{$gt:30}})`”.

Currently, there are several websites specifically designed for SQL-to-NoSQL conversion based on the syntax mapping between SQL and NoSQL [Site24x7(2023)], [JavaInUse(2023)], as well as GitHub projects [vincentrussell(2024)]. We utilized an open-source project from GitHub to transform the test set of the TEND dataset (which includes NoSQL queries and their corresponding SQL queries), resulting in the SQL-to-NoSQL by Grammar presented in Table I (**Grammar Converter**) and Table II. Table I displays the accuracy of the converted NoSQL queries, while Table II showcases some representative examples.

	Query-based		
Method	EM	QSM	QFC
SMART (Ours)	23.82%	63.21%	75.60%
Grammar Converter	0.00%	17.12%	23.24%
LLM Converter w. SQL Schema	10.09%	58.05%	64.90%
LLM Converter w/o. SQL Schema	7.71%	57.80%	63.35%

(a) Query-based Results

	Execution-based		
Method	EX	EFM	EVM
SMART (Ours)	65.08%	87.21%	72.29%
Grammar Converter	10.81%	70.45%	67.57%
LLM Converter w. SQL Schema	44.76%	71.57%	73.55%
LLM Converter w/o. SQL Schema	47.68%	69.19%	73.80%

(b) Execution-based Results

TABLE I: Experimental Results of SQL-to-NoSQL.

From the experimental results presented in Table I, we

Easy Find Query	
SQL	SELECT Address FROM Restaurant WHERE ResName = ”Subway”;
Target NoSQL	db.Restaurant.find({ ”ResName”: ”Subway” }, { ”Address”: 1, ”_id”: 0 });
Converted NoSQL	db.Restaurant.find({ ”ResName” : ”Subway” },{ ”Address”: 1 });
Medium Find Query	
SQL	SELECT last_name FROM staff WHERE email_address LIKE ”%wrau%”
Target NoSQL	db.Staff.find({ email_address: { $regex: ”wrau”, $options: ”i” } }, { last_name: 1, _id: 0 });
Converted NoSQL	db.staff.find({ ”email_address”: { ”$regex”: ”^.*wrau.*$” } } , { ”_id”: 0, ”last_name”: 1 })
Hard Find Query	
SQL	SELECT T2.Fname , T2.Lname FROM COURSE AS T1 JOIN FACULTY AS T2 ON T1.Instructor = T2.FacID WHERE T1.CName = ”COMPUTER LITERACY”;
Target NoSQL	db.Faculty.find({ ”Course.CName”: ”COMPUTER LITERACY” }, { ”Fname”: 1, ”Lname”: 1, ”_id”: 0 });
Converted NoSQL	db.COURSE.aggregate([{ ”$match”: { ”CName”: ”COMPUTER LITERACY” } },{ ”$lookup”: { ”from”: ”FACULTY”, ”let”: { ”instructor”: ”$Instructor” }, ”pipeline”: [{ ”$match”: { ”$expr”: { ”$eq”: [”$$instructor”, ”$FacID”] } } }], ”as”: ”T2” } },{ ”$unwind”: { ”path”: ”$T2”, ”preserveNullAndEmptyArrays”: false } },{ ”$project”: { ”_id”: 0, ”T2.Fname”: 1, ”T2.Lname”: 1 } }])
Easy Aggregate Query	
SQL	SELECT Nationality FROM pilot GROUP BY Nationality ORDER BY COUNT(*) DESC LIMIT 1
Target NoSQL	db.pilot.aggregate([{ $group: { _id: ”$Nationality”, count: { $sum: 1 } } }, { $sort: { count: -1 } }, { $limit: 1 }, { $project: { _id: 0, Nationality: ”$_id” } }]);
Converted NoSQL	db.pilot.aggregate([{ ”$group”: { ”_id”: ”$Nationality” } },{ ”$sort”: { ”count”: -1 } },{ ”$limit”: 1 },{ ”$project”: { ”Nationality”: ”$_id”, ”_id”: 0 } }])
Medium Aggregate Query	
SQL	SELECT company , main_industry FROM company WHERE company_id NOT IN (SELECT company_id FROM station_company)
Target NoSQL	db.company.aggregate([{ $lookup: { from: ”gas_station”, localField: ”Company_ID”, foreignField: ”station_company.Company_ID”, as: ”Docs1” } }, { $match: { Docs1: { $size: 0 } } }, { $project: { Company: 1, Main_Industry: 1, _id: 0 } }]);
Converted NoSQL	Converting Error
Hard Aggregate Query	
SQL	SELECT T1.first_name , T1.last_name , T1.employee_id , T4.country_name FROM employees AS T1 JOIN departments AS T2 ON T1.department_id = T2.department_id JOIN locations AS T3 ON T2.location_id = T3.location_id JOIN countries AS T4 ON T3.country_id = T4.country_id
Target NoSQL	db.departments.aggregate([{ $unwind: ”$employees” }, { $lookup: { from: ”regions”, let: { location_id: ”$LOCATION_ID” }, pipeline: [{ $unwind: ”$countries” }, { $unwind: ”$countries.locations” }, { $match: { $expr: { $eq: [”$countries.locations.LOCATION_ID”, ”$$location_id”] } } }, { $project: { COUNTRY_NAME: ”$countries.COUNTRY_NAME” } }], as: ”Docs1” } }, { $unwind: ”$Docs1” }, { $project: { FIRST_NAME: ”$employees.FIRST_NAME”, LAST_NAME: ”$employees.LAST_NAME”, EMPLOYEE_ID: ”$employees.EMPLOYEE_ID”, COUNTRY_NAME: ”$Docs1.COUNTRY_NAME”, _id: 0 } }]);
Converted NoSQL	db.employees.aggregate([{ ”$lookup”: { ”from”: ”departments”, ”let”: { ”department_id”: ”$department_id” }, ”pipeline”: [{ ”$match”: { ”$expr”: { ”$eq”: [”$$department_id”, ”$department_id”] } } }], ”as”: ”T2” } },{ ”$unwind”: { ”path”: ”$T2”, ”preserveNullAndEmptyArrays”: false } },{ ”$lookup”: { ”from”: ”locations”, ”let”: { ”t2_location_id”: ”$T2.location_id” }, ”pipeline”: [{ ”$match”: { ”$expr”: { ”$eq”: [”$$t2_location_id”, ”$location_id”] } } }], ”as”: ”T3” } },{ ”$unwind”: { ”path”: ”$T3”, ”preserveNullAndEmptyArrays”: false } },{ ”$lookup”: { ”from”: ”countries”, ”let”: { ”t3_country_id”: ”$T3.country_id” }, ”pipeline”: [{ ”$match”: { ”$expr”: { ”$eq”: [”$$t3_country_id”, ”$country_id”] } } }], ”as”: ”T4” } },{ ”$unwind”: { ”path”: ”$T4”, ”preserveNullAndEmptyArrays”: false } },{ ”$project”: { ”_id”: 0, ”first_name”: 1, ”last_name”: 1, ”employee_id”: 1, ”T4.country_name”: 1 } }])

TABLE II: Samples of converting SQL to NoSQL through grammatical mapping relationships using SQL-to-NoSQL converter [vincentrussell(2024)]. (Errors are marked with red colors)

have observed that direct conversion based on grammatical rules is not feasible. This is fundamentally because such an approach entirely overlooks the significant differences between relational databases and NoSQL databases, culminating in a mere 10.09% accuracy rate on the most crucial metric — EX. This level of performance is far from sufficient to meet the demands of practical applications. Concurrently, we have analyzed the error samples from the Grammar Converter, with several representative examples showcased in Tablee II. Based on the displayed examples, it is evident that the method of directly converting SQL queries into NoSQL queries according to grammatical mapping relationships can only be correct when the converted NoSQL query is a simple “find” query. However, this method becomes unfeasible when the NoSQL query to be converted becomes complex. Currently, large language models (LLMs) are revolutionizing and optimizing solutions across various research domains. The SQL-to-NoSQL conversion process might benefit from leveraging LLMs’ understanding of both programming languages. We attempted to provide an LLM (such as `Deepseek-v3`) with ground truth SQL queries and a MongoDB database schema, instructing it to translate the SQL queries into corresponding MongoDB queries. The experimental setup was as follows: (i) Providing an SQL query, the complete database schema corresponding to the SQL query, and the complete MongoDB database schema (LLM Converter w. SQL Schema); (ii) Not providing the SQL database schema (LLM Converter w/o. SQL Schema). The experimental results obtained are also presented in Table I.

Based on the results presented in Table I, we have found that leveraging LLMs for SQL-to-NoSQL conversion demonstrates a certain level of feasibility. By furnishing the LLM with precise SQL queries, the corresponding database schema for the SQL queries, and the NoSQL database schema, we achieved an execution accuracy of 44.76%. This figure significantly surpasses the 10.81% accuracy rate attained by the Grammar Converter. However, it is important to note that in practical applications, obtaining accurate SQL queries is not always accurate. Incorrect SQL queries may lead to error accumulation, potentially resulting in a significant decline in accuracy. Therefore, to pursue higher accuracy, it may be necessary to design a more complex LLM framework, which represents a promising direction for further research.

III. Prompt Template

A. *SLM-based Schema Prediction*

```
System: You are now the MongoDB natural language
    interface, responsible for converting user
    input natural language queries into MongoDB
    query statements based on the MongoDB
    Collection and their Fields, and parsing the
    features according to user requirements.

Human: # Given the natural language query, please
     predict the fields used in the query.

## Natural Language Query: `<NLQ>`

## MongoDB Collection and their Fields
### Collection: <collection>
- <field>: <field_type>
...
- <field>: (Array)
  - <sub_field>: <field_type>
...

Assistant:
```

B. *SLM-based Query Generation*

```
System: You are now the MongoDB natural language
    interface, responsible for converting user
    input natural language queries into MongoDB
    query statements based on the MongoDB
    collections and their fields.

Human: # Given the MongoDB collections and their
    fields and natural language query, please
    generate final MongoDB query.

## Natural Language Query: `<NLQ>`

## MongoDB Collection and their Fields
### Collection: <collection>
- <field>: <field_type>
...
- <field>: (Array)
  - <sub_field>: <field_type>
...

Assistant:
```

C. *Predicted Schema-driven and Retrieved Example-driven Query Refinement*

~~~
## Query Transformation Reference Examples
### <e_id>. Example <e_id>
#### Natural Language Query
- `<e_NLQ>`
#### MongoDB Collections Used in MongoDB Query
  - `<collections>`
#### MongoDB Fields Used in MongoDB Query
  - `<fields>`
#### Renamed Fields Used in MongoDB Query
  - `<rename_fields>`
#### Fields shown in Execution Document
  - `<target_fields>`
#### MongoDB Query
```javascript
<MQL>
~~~
...

## MongoDB collections and their fields
### Collection: <collection>
- <field>: <field_type>
...
- <field>: (Array)
  - <sub_field>: <field_type>
...

## Natural Language Query
  - `<NLQ>`
## Original MongoDB Query
```javascript
<MQL>
```

## MongoDB Collections Used in MongoDB Query
  - `<collections>`
## MongoDB Fields Used in MongoDB Query
  - `<fields>`
## Renamed Fields Used in MongoDB Query
  - `<rename_fields>`
## Fields shown in Execution Document
  - `<target_fields>`

# Given MongoDB collections and their fields, a
    natural language query, query transformation
    reference examples, and the original MongoDB
    query, please perform the following actions:
1. Analyze whether the original MongoDB query
    needs adjustment based on the natural
    language query and the MongoDB collections
    and their fields:
  - If adjustments are needed, analyze the
      natural language query based on the MongoDB
       collections and their fields (only adjust
      if necessary);
  - If no adjustments are needed, retain the
      original MongoDB query and proceed directly
       to step three for output;
2. Adjust the original MongoDB query based on the
     query transformation reference examples and
    the analysis from step one, then proceed to
    step three for output;
```
```

~~~
3. Output the final MongoDB query in the
    following format:
```javascript
db.collection.aggregate([pipeline]); / db.
    collection.find({[filter]}, {[projection]});
~~~

A: Let's think step by step!
```

## D. Execution Result-based Query Optimization

```
Reference Exampels:
<e_id>. Example <e_id>
Natural Language Query
  - `<NLQ>`
Fields Shown in the Execution Results
  - `<target_fields>`
Gold MongoDB Query
```javascript
<MQL>
```
#### Gold Execution Resutls
```json
<exec_results>
```

## MongoDB collections and their fields
### Collection: <collection>
- <field>: <field_type>
...
- <field>: (Array)
  - <sub_field>: <field_type>
...

## Natural Language Query
   - `<NLQ>`

## Fields Shown in the Execution Results
   - `<target_fields>`

## Original MongoDB Query
```javascript
<MQL>
```

### Execution Results
```json
<exec_results>
```

# Given MongoDB collections and their fields, a
    natural language query, the original MongoDB
    query, the execution result of the original
    MongoDB query, and reference examples of
    queries and results, please perform the
    following actions:
1. Analyze the reference examples of queries and
    results to summarize the following
    information:
  - How each stage of the MongoDB query should
      operate when handling various natural
      language queries;
  - What key names need to be included in the
      final documents when handling various
      natural language queries;
2. Based on the MongoDB collections and their
    fields and the natural language query,
    analyze the original MongoDB query and its
    corresponding execution result (pay special
    attention to the operations in the query and
    the key names in the result documents);
3. Combine the analysis of the reference examples
     from step one and the analysis of the
    original MongoDB query and its execution
    result from step two to determine whether the
     original MongoDB query needs adjustment (
    such as the operations in the MongoDB query
    and the key names in the result documents);
  - If adjustments are needed, adjust the original
      MongoDB query based on the analysis from
     steps one and two;
  - If no adjustments are needed, retain the
     original MongoDB query;
4. Output the final MongoDB query in the
    following format:
```javascript
db.collection.aggregate([<pipeline>]); / db.
    collection.find({<filter>}, {<projection>});
```

A: Let's think step by step!
```

## References

[Finegan-Dollak et al.(2018)] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving Text-to-SQL Evaluation Methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, Melbourne, Australia, 351–360. https://doi.org/10.18653/v1/P18-1033

[JavaInUse(2023)] JavaInUse. 2023. SQL to MongoDB Query Converter. https://www.javainuse.com/sql2mongo. Accessed: 2025-1-11.

[Pourreza and Rafiei(2023)] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 36339–36348. https://proceedings.neurips.cc/paper_files/paper/2023/file/72223cc66f63ca1aa59edaec1b3670e6-Paper-Conference.pdf

[Site24x7(2023)] Site24x7. 2023. SQL to MongoDB Query Converter. https://www.site24x7.com/tools/sql-to-mongodb.html. Accessed: 2025-1-11.

[vincentrussell(2024)] vincentrussell. 2024. SQL to MongoDB Query Converter. https://github.com/vincentrussell/sql-to-mongo-db-query-converter.git. Accessed: 2025-1-16.

[Wang et al.(2020)] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 7567–7578. https://doi.org/10.18653/v1/2020.acl-main.677

[Yu et al.(2018)] 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. 2018. 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*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3911–3921. https://doi.org/10.18653/v1/D18-1425
```