QueryCraft: A Natural Language-Driven NoSQL Database Querying System Powered by Large Language Models

Jinwei Lu*
jinwei.lu@polyu.edu.hk
The Hong Kong Polytechnic University
Hong Kong, China

Zhiqian Qin*
zhiqianqin1206@gmail.com
The Hong Kong Polytechnic University
Hong Kong, China

Chen Zhang
jason-c.zhang@polyu.edu.hk
The Hong Kong Polytechnic University
Hong Kong, China

Haodi Zhang
hdzhang@szu.edu.cn
Shenzhen University
Shenzhen, China

Yuanfeng Song
songyf@outlook.com
WeBank
Shenzhen, China

Raymond Chi-Wing Wong
raywong@cse.ust.hk
The Hong Kong University of Science and Technology
Hong Kong, China

ABSTRACT

QueryCraft is an interactive MongoDB interface for non-relational NoSQL document databases through which users ask natural language questions, inspect generated aggregation pipelines, and execute queries without manually writing MongoDB code. The interface includes a nested schema browser, an inspectable program view, a formatted results panel, and a local history panel for reviewing previous attempts. The text-to-NoSQL generator is replaceable; this implementation uses the Schema-as-Data Grounding (SAG) solver from the Text-to-NoSQL Dataset (TEND) release, with schema grounding, guarded decoding, execution repair, and result consistency. The demo exposes generated pipelines, execution results, runtime errors, and solver metadata, giving users and researchers a concrete view of successful queries and failure cases.

PVLDB Reference Format:
Jinwei Lu, Zhiqian Qin, Chen Zhang, Haodi Zhang, Yuanfeng Song, Raymond Chi-Wing Wong. QueryCraft: A Natural Language-Driven NoSQL Database Querying System.... PVLDB, 19(12): 4754 - 4757, 2026.
doi:10.14778/3827998.3828114

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Jinwei-Lu/Text-to-NoSQL.git.

1 INTRODUCTION

Recent progress in Text-to-NoSQL. NoSQL databases, an umbrella category covering document, graph, key-value, and wide-column systems, have become widely adopted due to their flexibility in handling large-scale, semi-structured data [12]. Recent advances in natural language processing have stimulated interest in natural language database interfaces [4, 8–11, 13], while Text-to-NoSQL has drawn explicit attention through MultiTEND, a 2025 multilingual benchmark built on the Text-to-NoSQL Dataset (TEND) [6]. MongoDB interfaces must resolve nested document paths, heterogeneous field types, and dependencies across aggregation stages before they can produce executable queries (Figure 1). These requirements make it difficult for end users to identify the correct fields and construct semantically valid queries.

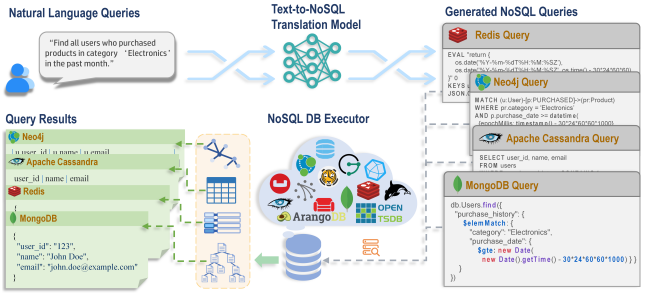

Figure 1: The text-to-NoSQL task maps natural language questions to NoSQL queries that may use nested document fields and aggregation pipelines.

Demo focus: inspectable MongoDB query generation. Rather than reporting only benchmark accuracy, this demo shows schema context, generated pipelines, execution output, and failure messages in one interface. It focuses on document database tasks that existing Structured Query Language (SQL) query-generation demos usually omit, including nested schema browsing and debugging across aggregation stages. Because the generator sits behind one service boundary, different Text-to-NoSQL methods can be evaluated with the same interface.

Why Text-to-SQL cannot replace Text-to-NoSQL. Text-to-SQL methods have progressed from neural semantic parsers to systems built around large language models (LLMs) [1, 2, 5, 7, 14], but they do not transfer directly to MongoDB. SQL assumes stable table and column pairs, whereas MongoDB queries often refer to paths inside nested objects and arrays. MongoDB aggregation pipelines also pass aliases across stages, and SQL lacks native constructs such as `$unwind` for nested arrays. As a result, SQL-to-NoSQL conversion

*Jinwei Lu and Zhiqian Qin contributed equally to this work.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828114

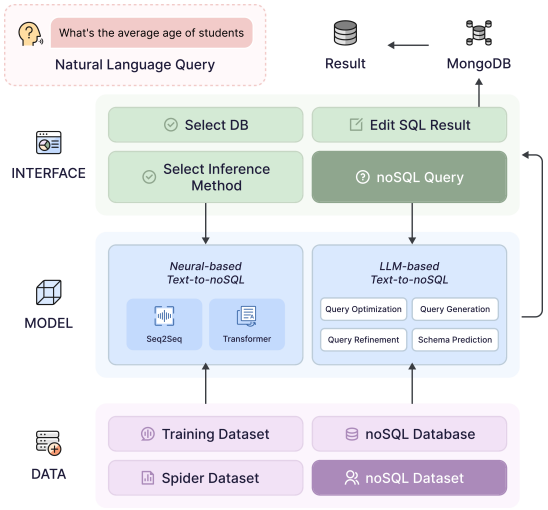

Figure 2: An overview of the QueryCraft architecture, organized into three layers: Interface, Model, and Data.

can drop required stages or produce lower execution accuracy than direct Text-to-NoSQL generation in prior studies [3].

QueryCraft demo. In QueryCraft, users browse a hierarchical schema, generate and inspect a MongoDB aggregation pipeline, execute it when MongoDB is available, and revisit prior queries through a history panel. The current prototype targets MongoDB as a representative document database, with portability discussed in Section 4. The generation backend uses the Schema-as-Data Grounding (SAG) solver, which follows the Text-to-NoSQL direction established by Lu et al. [3] while exposing additional runtime metadata for the demo.

Sections 2 to 4 describe the architecture, demonstration scenarios, and implementation.

2 SYSTEM OVERVIEW

QueryCraft has three layers: a web interface, a Text-to-NoSQL generator, and MongoDB data sources. The workflow centers on schema inspection, program generation, execution, and review.

2.1 User Interaction Workflow

Step 1 (Database selection and schema browsing). Users select a MongoDB demo database and inspect collections, nested fields, and field types in the schema browser. This view reduces ambiguity in document schemas by expanding objects and arrays directly in the interface.

Step 2 (Natural language query input and generation). After the user enters a natural language question, QueryCraft generates a MongoDB aggregation pipeline and displays it in a syntax-highlighted view.

Step 3 (Execution and result visualization). The user can request execution against MongoDB and view the formatted result or error message. The browser stores the attempt for later replay.

Table 1: Current solver and baseline results on the TEND native MongoDB release. Bounded column-tolerant execution accuracy (EXC) and graded result-set F_1 (EXF_1) are reported as percentages.

Method	EXC	EXF_1
Direct LLM	28.6%	29.4%
Data-rich direct LLM	28.9%	29.8%
SQL pivot workflow	25.6%	26.2%
Reasoning and acting agent	30.8%	34.6%
TEND SAG full solver	**35.8%**	**40.5%**

Step 4 (Review through history). The local history panel lists recent questions and generated programs so users can compare attempts.

2.2 Design Choices for MongoDB Queries

Nested schema browsing for document databases. MongoDB schemas are hierarchical and often include arrays of embedded documents, so users must identify paths such as `a.b.c` before checking a generated pipeline. Unlike SQL's flat table and column schemas, which can often be listed in a single view, document databases may place the same logical attribute at multiple nesting levels. QueryCraft expands nested objects and arrays and displays field types beside each path.

Inspectable program view. QueryCraft displays the generated MongoDB program with syntax highlighting and supports copying the generated MongoDB Query Language (MQL) program for manual inspection. Users can check stage order, aliases, and projections, where aggregation pipelines often fail. Because downstream stages may consume aliases created upstream, one missing or misordered stage can invalidate the pipeline.

Execution feedback for debugging. When a query fails or returns unexpected results, runtime messages identify errors such as unknown fields, invalid operators, and alias mismatches. QueryCraft shows these traces beside the generated program and solver metadata, including the selected collection, pipeline, repair rounds, consistency cluster size, and diagnostic metadata.

Query review through history. Text-to-program generators often require revision, especially on complex schemas. The browser stores recent records containing the natural language question, generated program, result type, and elapsed time so users can review previous queries.

2.3 Text-to-NoSQL Backend

The user interface calls a text-to-NoSQL generator through a fixed interface, so the backend can use direct prompting or a modular pipeline. The current implementation uses the SAG solver: for each database it builds a grounding index, renders a schema path card, applies path and value guards, repairs candidates with execution feedback, and chooses among multiple attempts by result consistency. The public demo release contains 11 native MongoDB databases and 1,210 natural language to MQL tasks.

Why not cascade through Text-to-SQL? Table 1 compares representative methods on the TEND benchmark to motivate the choice

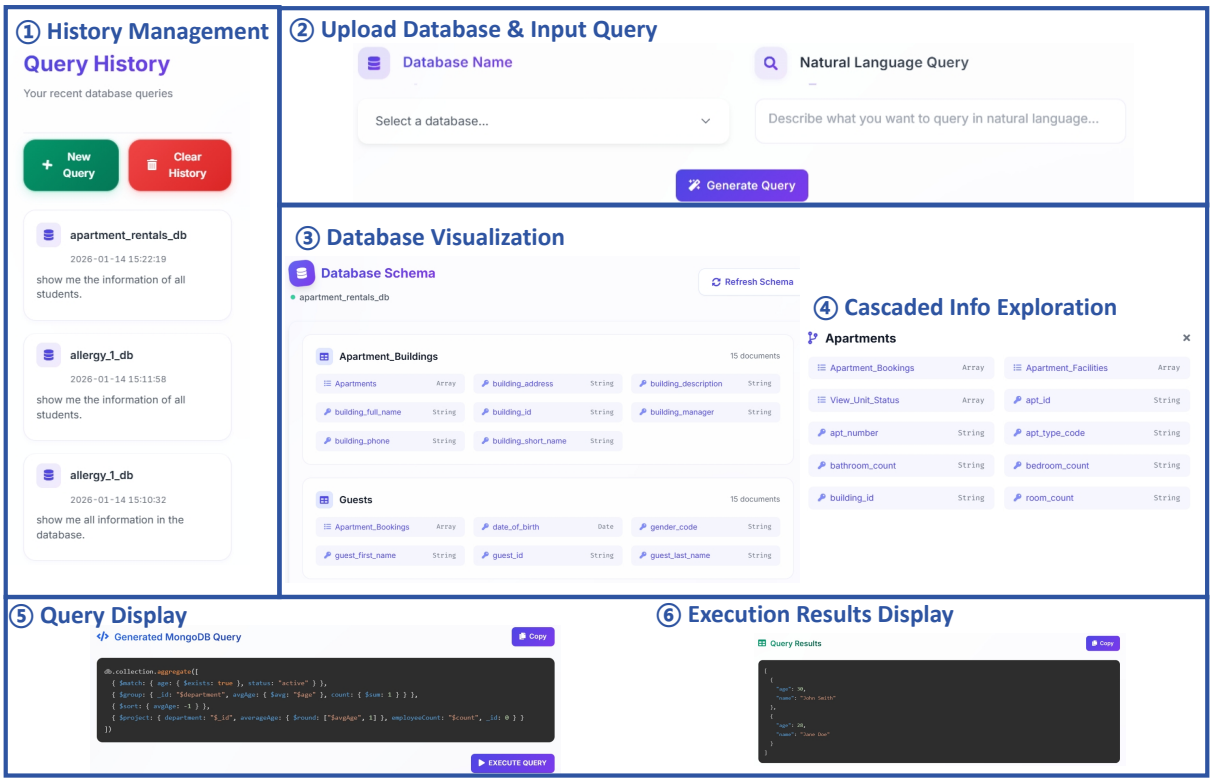

Figure 3: QueryCraft user interface overview. ① Local history panel for query replay. ② Database selection and natural language query input. ③ Schema browser displaying collections and field types. ④ Cascaded exploration of nested document fields. ⑤ Generated query display. ⑥ Visualization of query execution results.

of backend. The SQL pivot workflow first asks an LLM for a SQL sketch and then translates that sketch to MQL. One reason is that SQL lacks native constructs such as `$unwind` over nested arrays or alias propagation across aggregation stages, so conversion can drop stages or produce semantically incorrect pipelines. The TEND SAG full solver improves over direct LLM prompting and the SQL pivot workflow, motivating its adoption as the default backend while keeping the interface independent of the generator implementation.

The solver also exposes intermediate metadata, including the selected collection, generated pipeline, repair rounds, sample count, consistency cluster size, final violation status, execution status, and diagnostic metadata. QueryCraft displays this metadata beside the program and execution output to show why a generated query succeeded or failed (Section 3, Scenario 3).

3 DEMONSTRATION SCENARIOS

The demo contains three scenarios: a basic query, nested schema exploration, and failure diagnosis. Figure 3 labels the history panel, query input, schema browser, generated program, and results panel.

Scenario 1: Basic query execution. A user selects a MongoDB demo database and browses available collections and fields in the schema (Figure 3, regions ② and ③). The user then enters a natural language query and inspects the syntax-highlighted MongoDB pipeline (Figure 3, region ⑤). The user requests execution and views the formatted results (Figure 3, region ⑥). For example, given "How many customers live in the city of *Lake Geovannyton*?", the system generates an aggregation pipeline that includes `$unwind`, `$lookup`, `$match`, `$group`, and `$project`, and returns `[{ "count": 3 }]`. This scenario establishes the query path used in the later scenarios.

Scenario 2: Nested schema exploration and complex pipeline generation. The second scenario uses a database whose schema contains deeply nested arrays of embedded documents, such as a `students` collection where each document embeds `test_results`, which in turn contain `scores`. Using the cascaded schema explorer (Figure 3, region ④), the user expands nested fields to find the target completion-date path and verify field types. The user then enters a natural language question such as "List the completion dates of tests that received a failing result." QueryCraft generates a multi-stage pipeline involving successive `$unwind` operations over nested arrays, an optional `$lookup` for cross-collection joins, and a `$project` stage that extracts the specified nested path. Before execution, the user checks whether each `$match` references fields or aliases created by earlier stages (Figure 3, region ⑤).

Scenario 3: Failure diagnosis and iterative refinement. Complex schemas can cause Text-to-NoSQL generators to choose the wrong path, omit an `$unwind`, or reuse an alias incorrectly. In this scenario, the system generates a pipeline that fails at execution time.

For example, a `$match` stage may reference a nested path that was not unwound earlier, or a `$group` alias may not match a subsequent `$project`. QueryCraft surfaces the runtime error message alongside the generated pipeline and the solver metadata. The user can compare the current query with previous successful queries via the history panel (Figure 3, region ①), identify the discrepancy, copy the generated MQL for inspection, or rephrase the natural language question for a new generation attempt. The scenario shows the diagnostic path: inspect the error, compare earlier attempts, inspect the pipeline, or rephrase the question. During the presentation, this scenario presents a failed program and a revised attempt together, so the diagnostic value comes from visible evidence rather than an opaque model explanation.

4 IMPLEMENTATION

QueryCraft is implemented as a browser-based demonstration system with a Flask service layer between the user interface, the TEND native MongoDB release, and the text-to-NoSQL generator. The service handles database discovery, example retrieval, schema extraction, query generation, and optional execution through structured web endpoints. Generation remains a separate module: it receives the natural language question and schema context and returns an inspectable MongoDB program.

For each selected database, QueryCraft builds a shared schema representation with collection names, nested paths, field types, document counts, and sample documents. The same representation drives the schema browser and the generator context. The browser stores up to 12 recent runs, including the question, generated program, result type, and elapsed time.

The current prototype instantiates this interface with the SAG solver, while the same boundary can support direct LLM prompting or other modular Text-to-NoSQL pipelines. Users may execute the generated program; the backend validates the database, builds context from the shared schema representation, runs the query when requested, and returns formatted results or structured errors. This separation keeps solver reasoning apart from schema exploration, program inspection, execution feedback, and history replay.

The frontend is a custom single-page browser interface with four coordinated views: database and query input, hierarchical schema browsing, generated-program inspection, and result plus history review. This organization supports MongoDB query debugging by keeping nested fields, aggregation-stage dependencies, execution output, and previous attempts in one workflow. Each history record preserves the question, generated MQL, execution status, result summary, and elapsed time, which also supports controlled comparisons among generators.

Extensibility to other NoSQL models. The current system targets MongoDB as a representative document-oriented NoSQL database. Its schema loading, query rendering, parsing, execution, and solver prompts assume MongoDB documents and aggregation pipelines. Extending QueryCraft to similar document databases, such as Couchbase or Amazon DocumentDB, would require adapter work around the same interaction workflow. Supporting key-value, wide-column, or graph databases would require new schema abstractions, query representations, and training or prompting resources. Portability is therefore treated as an interface contract rather than a claim that MongoDB-specific query code can be reused unchanged.

5 CONCLUSION

QueryCraft lets users ask MongoDB questions in natural language, inspect the generated aggregation pipeline, execute it, and revisit prior attempts. By placing schema paths, pipeline stages, execution feedback, and previous queries in one interface, the system targets failure modes that conventional Text-to-SQL demonstrations do not cover. Future work will measure whether the interface improves query correctness in user studies and will adapt the generator interface to other NoSQL data models. The presentation highlights whether schema paths exist, stages preserve aliases, and execution feedback matches intent, giving a compact basis for comparing future Text-to-NoSQL generators. The demonstration video is available at https://youtu.be/0BtfCyD_edE.

ACKNOWLEDGMENTS

Yuanfeng Song is the corresponding author.

REFERENCES

[1] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 4524–4535.

[2] George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *The VLDB Journal* 32, 4 (2023), 905–936.

[3] Jinwei Lu, Jiawei Lu, Chen Zhang, Zhiqian Qin, Haodi Zhang, Yuanfeng Song, and Raymond Chi-Wing Wong. 2026. Bridging the Gap: Enabling Natural Language Queries for NoSQL Databases through Text-to-NoSQL Translation. *arXiv preprint arXiv:2502.11201* (2026), 15.

[4] Jinwei Lu, Yuanfeng Song, Chen Zhang, and Raymond Chi-Wing Wong. 2026. MultiVis-Agent: A Multi-Agent Framework with Logic Rules for Reliable and Comprehensive Cross-Modal Data Visualization. *Proceedings of the ACM on Management of Data* 4, 1 (apr 2026), 1–25.

[5] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. *Advances in Neural Information Processing Systems* 36 (2023), 36339–36348.

[6] Zhiqian Qin, Yuanfeng Song, Jinwei Lu, Yuanwei Song, Shuaimin Li, and Chen Jason Zhang. 2025. MultiTEND: A Multilingual Benchmark for Natural Language to NoSQL Query Translation. In *Findings of the Association for Computational Linguistics: ACL 2025*. 24632–24657.

[7] Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X. Sean Wang. 2024. PURPLE: Making a Large Language Model a Better SQL Writer. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 15–28.

[8] Yuanfeng Song, Jinwei Lu, and Raymond Chi-Wing Wong. 2026. CoVis: Neural and LLM-Driven Multi-Turn Interactions for Conversational Text-to-Visualization Generation. *The VLDB Journal* 35, 1, Article 3 (2026), 25 pages.

[9] Yuanfeng Song, Raymond Chi-Wing Wong, and Xuefang Zhao. 2024. Speech-to-SQL: toward speech-driven SQL query generation from natural language question. *The VLDB Journal* 33, 4 (2024), 1179–1201.

[10] Yuanfeng Song, Raymond Chi-Wing Wong, Xuefang Zhao, and Di Jiang. 2022. VoiceQuerySystem: A Voice-driven Database Querying System Using Natural Language Questions. In *Proceedings of the 2022 International Conference on Management of Data*. 2385–2388.

[11] Yuanfeng Song, Xuefang Zhao, Raymond Chi-Wing Wong, and Di Jiang. 2022. RGVisNet: A Hybrid Retrieval-Generation Neural Framework Towards Automatic Data Visualization Generation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1646–1655.

[12] Michael Stonebraker. 2010. SQL databases v. NoSQL databases. *Commun. ACM* 53, 4 (2010), 10–11.

[13] Weixu Zhang, Yifei Wang, Yuanfeng Song, Victor Junqiu Wei, Yuxing Tian, Yiyan Qi, Jonathan H. Chan, Raymond Chi-Wing Wong, and Haiqin Yang. 2024. Natural Language Interfaces for Tabular Data Querying and Visualization: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 36, 11 (2024), 6699–6718.

[14] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv preprint arXiv:1709.00103* (2017), 12.