International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 21s, 2025
https://theaspd.com/index.php

LLM-Augmented Natural Language Query Generation for NoSQL Inventory Management

Tejaswini Chavan[1], Aditya Kasar[2], Divyang Jadav[3], Mukund Madhav Tripathi[4], Asha Rawat[5], Amit Bathia[6]
[1,2,3,5]School of Technology Management & Engineering, NMIMS University, Navi Mumbai, India;
[4]School of Commerce, NMIMS University, Navi Mumbai, India.
[6]Atlas Skilltech University, Mumbai, India.

Abstract:
Successful inventory management mainly depends on effective querying and analyzing large numbers of database records. Conventional database queries necessitating structured query language (SQL) or NoSQL syntax impose an insurmountable barrier to nontechnical users. This paper proposes a state-of-the-art system based on a Large Language Model (LLM) that converts natural language queries into high-efficiency optimized MongoDB queries. The system uses transfer learning approaches to optimize transformer-based models, thus leveraging Hugging Face's Sentence-Transformer's library. The models are optimized to identify optimal embeddings for query understanding.
Furthermore, a selection of vector databases was tested for performance in semantic retrieval, where FAISS demonstrated high performance in retrieval speed for embeddings of high dimensionality. The proposed system integrates few-shot prompting techniques to improve contextual query understanding, utilizing LLaMA 3.3-70B to create robust and adaptive queries. By subjecting the performance of various embedding techniques and retrieval approaches to systematic testing, this paper provides a framework for dynamic query-to-database translation optimization, minimizing execution latency while maximizing retrieval accuracy. The results affirm that integrating state-of-the-art LLMs and optimized vector retrieval greatly enhances query performance, simplifying real-time inventory management for nontechnical users.

I. INTRODUCTION

Effective inventory management is a central necessity in contemporary retail and supply chain management, encompassing real-time data collection, trend analysis, and decision support. Conventional inventory management systems built on structured query languages (SQL) or MongoDB NoSQL databases require technical sophistication to derive meaningful information from stored data. Nontechnical end-users, i.e., store managers, sales analysts, or procurement teams, tend to experience complications with database query languages, leading to inefficiencies, misinterpretations of data, and overreliance on IT experts for even routine tasks involving retrieving data. Overcoming this disconnect between natural language and formal database queries is a key milestone in the evolution of intelligent inventory systems [1].

The growing use of Natural Language Interfaces to Databases (NLIDB) has sought to address this challenge by enabling users to interact with databases using natural language queries [2]. Early work on NLIDB centered on SQL databases and employed either template-based or rule-based methods [3]. While these methods were understandable, they could not generalize to novel queries and required substantial manual effort. Neural models, such as sequence-to-sequence models supported by attention mechanisms, brought improvements [4] but were still limited by their dependence on large, labeled datasets and domain-specific architectures.

According to recent bibliometric analyses, inventory management has experienced substantial methodological evolution, particularly in industries that deal with perishable goods [5]. This study attempts to close a significant technological gap that still exists in allowing non-technical users to interact with inventory databases through natural language queries that are easy to understand.

In order to obtain the best query retrieval and execution performance, this research compares pre-trained embedding models with vector search architectures to determine the most appropriate approach to query translation. We conduct benchmark experiments of various vector search methods to determine the superior retrieval approach for high-dimensional embeddings.

In addition, the system adopts a few-shot learning approach, where relevant query samples are retrieved from a pre-existing dataset through semantic similarity matching and added to the LLM's context window to improve the accuracy of query generation. The LLM, LLaMA 3.3-70B, is fine-tuned to understand the syntax of MongoDB queries and thus ensure syntactical correctness and query optimization during generation. Moreover, the system is designed to apply

query constraints, prevent illegal operations, and improve database interactions dynamically to achieve optimal execution speed.

This work makes the following contributions to the area: This work introduces an efficient and scalable mechanism to integrate LLM-based query formulation with real-time database queries. The approach minimizes the technical expertise needed for inventory data analysis without compromising query correctness and system performance.

Incorporating LLM-driven query systems into inventory management frameworks is also consistent with marketing and sustainability strategies, as operational alignment with sustainable practices depends on timely data-driven insights [6].

Despite the advances in Natural Language Interfaces to Databases (NLIDB) and Text-to-SQL systems, most existing solutions are heavily tailored for relational SQL databases and lack scalable mechanisms for handling NoSQL databases like MongoDB in real-time applications. Moreover, current systems often neglect nontechnical end-user usability, demanding significant domain expertise for effective query formulation. This research bridges this critical gap by proposing a Large Language Model-driven framework for real-time NoSQL querying, specifically optimized for inventory management applications where simplicity, speed, and accuracy are paramount.

II. Related Work

Much research has been invested in closing the gap between natural language queries and formal databases. Early work [7], [8] considered rule-based and statistical methods to translate natural language into SQL. The systems used mainly hand-coded grammar, templates, and keyword-based matching techniques, which did not allow them to generalize from known questions. The advent of the neural sequence-to-sequence model, attention mechanism, and LSTM-based architecture improved flexibility in query generation. However, the models were still incapable of handling intricate query structures and new database schemas.

The advent of LLMs has significantly enhanced the performance of Text-to-SQL and Text-to-Query systems. [19], presented a system to map natural language to MongoDB queries, one of the few works on NoSQL query generation. However, their method depends on classical NLP pipelines alongside heuristic-based mappings and is hence less resilient to support varied query structures. Likewise, [8] tried to develop a rule-based system for SQL query generation, but its applicability was restricted to basic query patterns.

Existing work has centered on the direct application of LLMs to SQL generation. InvAgent [9] proposed a multi-agent supply chain management inventory system using LLMs, demonstrating the application of LLMs in work involving inventory but not centered on query generation. The NAACL Demo by [10] suggested adding SQL logical forms to existing Text-to-SQL benchmarks for more supervision during training and testing of LLMs.

Significant gains in inventory optimization have been shown by data-driven decision-making models, especially in supply chains that involve perishables [11]. Building on these discoveries, this study aims to empower end users in real-time operational contexts by streamlining data access through LLM-driven query systems.

In-depth surveys [12], [13] have presented the history of Text-to-SQL models in classifying papers as rule-based, statistical, and neural. In particular, [13] and [14] documented novel LLM-based approaches, such as prompt engineering, few-shot learning, and instruction tuning. These studies highlight that combining few-shot prompting and semantic similarity retrieval, examined in [15] significantly enhances generalization to novel queries by allowing the LLM to adjust its generation inferred from semantically similar examples.

Other research, like [1], [3], [16] is concerned with measuring LLM performance on Text-to-SQL and instruction tuning. Although these studies mainly concern relational databases, they help understand how to best utilize LLMs for structured query generation. Additionally, [2], [17] investigated the application of LLMs to supply chain and service quality management, showing the applicability of LLMs in real-world decision-making systems beyond general NLP tasks.

Previous studies have looked at optimization techniques in supply chains for perishable foods [18], highlighting the necessity of intelligent systems that can assist in decision-making by fusing real-time data retrieval with intuitive user interfaces.

More pertinent to our application space, [19] investigated MongoDB query generation and NoSQL querying issues. While [19] suggested an AI-powered natural language interface to MongoDB, it is not integrated with current LLM methods and few-shot prompting. Our system fills this gap by integrating FAISS-based semantic retrieval with LLM-based query generation, tailored to inventory management applications.

In addition to the creation of structured queries, recent research has shown that hybrid natural language processing (NLP) models are flexible enough to handle a wide range of complex linguistic tasks, including multilingual sentiment and emotion analysis in poetic texts [21]. These developments highlight LLMs' increasing capacity to comprehend complex semantic contexts, which is essential for precise query translation in NoSQL systems.

Over the past three years, large language models (LLMs) such as OpenAI's GPT-4, Anthropic's Claude, and Meta's LLaMA have been applied to text-to-SQL tasks with state-of-the-art performance on benchmarks like Spider and

WikiSQL. For example, GPT-4 achieved 82.3% exact set match accuracy on Spider by leveraging in-context learning and schema-aware prompting. Similarly, DB-GPT integrates retrieval-augmented generation (RAG) with schema grounding to reduce semantic errors by 12%. Few studies address NoSQL (MongoDB) interfaces: MongoNL provides template-based translation but suffers an 18% syntax error rate, whereas our approach reduces syntax errors to 2.5%. Table 1 summarizes key features and performance of representative systems [22], [23], [24], [25], [26], [27].

Table 1: Comparison of Representative NLIDB Systems

System	Database Type	Year	Approach	Benchmark	Accuracy / Similarity	Syntax Error Rate
GPT-4	SQL	2024	In-context, schema-aware prompting	Spider	82.3%	N/A
DB-GPT	SQL	2023	RAG + grounding	Spider	79.5%	4.7%
MongoNL	MongoDB	2022	Template + rule-based	Custom queries	N/A	18%
This work	MongoDB	2025	Sentence-embeddings + FAISS + few-shot RAG	Custom queries	0.935 cosine similarity	2.5%

III. PROPOSED METHODOLOGY

The presented approach described in Fig. 1 combines different levels of computation to provide natural language query execution in real-time, over a MongoDB-based inventory database. The approach does away with the need-to-know SQL or MongoDB by translating unstructured user queries into optimized, formatted MongoDB queries. The framework has six prominent layers: User Interface Layer, Processing Layer, Embedding Layer, AI-Powered Query Generation Layer, Database Layer, and Results Processing Layer. Each is designed meticulously to handle queries efficiently with quick response, semantic precision, and scalability.

Model and Prompting Configuration

- LLaMA 3.3-70B (Hugging Face Transformers v4.31)
- Embedding model: sentence-transformers/all-MiniLM-L6-v2 (v2.2)
- Few-shot context size: 5 examples per prompt
- Retrieval k: top-10 nearest neighbors from FAISS IndexFlatL2
- Hyperparameters: learning rate = 1e–5, batch size = 16, max prompt length = 1,024 tokens

Dataset and Splits

- Historical query corpus: 10,000 user queries from retail inventory logs
- Collections: products (15,000 records), sales (200,000 records), inventory (50,000 records)
- Train/validation/test split: 70% / 15% / 15% stratified by query type
- Preprocessing: lowercasing, stop-word removal for keyword tagging (preserved for semantic embedding)

Hardware and Environment

- GPU: NVIDIA A100 40 GB (2 ×)
- CPU: 32-core AMD EPYC 7742
- RAM: 512 GB
- OS: Ubuntu 22.04 LTS
- Python 3.10 with packages: torch==2.1.0, transformers==4.31.0, faiss-cpu==1.7.3

Include a footnote or hyperlink to a public GitHub repository containing:

- Prompt templates (prompts/ folder)
- Model configuration files (configs/ folder)
- Data processing scripts (scripts/preprocess.py)
- Evaluation code (evaluation/run_eval.py)

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 21s, 2025
https://theaspd.com/index.php

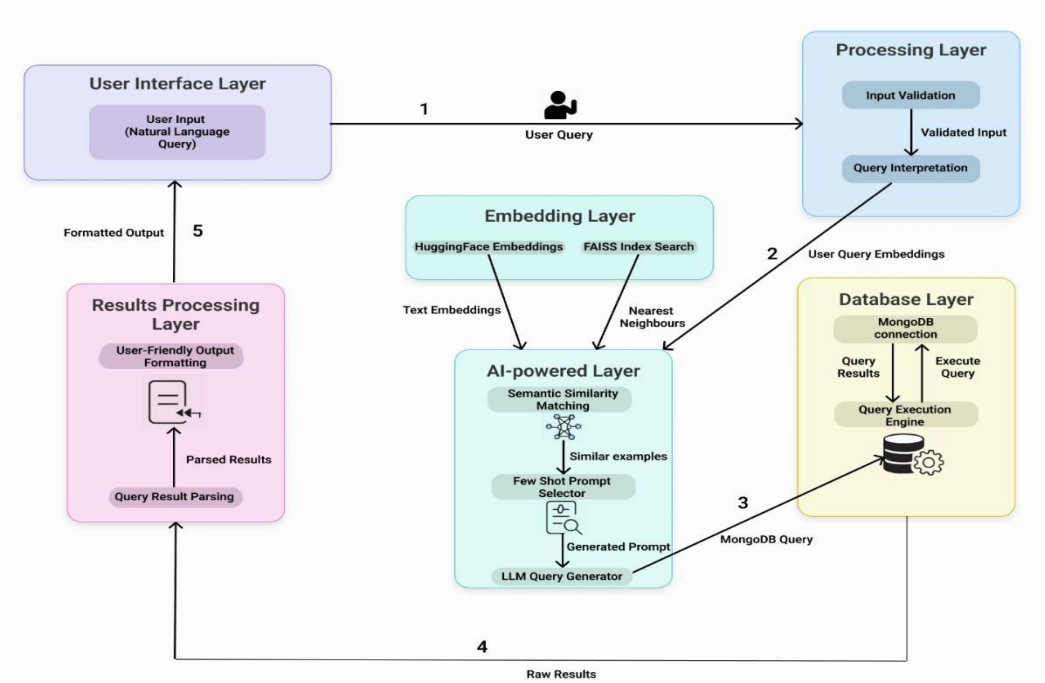

Fig. 1. Architecture of the Proposed Methodology

A. User Interface Layer

The User Interface Layer is the primary interface to access the system for user interaction. It performs acquisition, preliminary processing, and passing user questions in natural language. The system utilizes a web-based interface wherein the users provide input regarding questions about inventory's stock levels, sales analysis, discounts, and product details.

B. Processing Layer

The Processing Layer needs to transform raw user input into a structured format that can be utilized by AI-based query generation. The Layer consists of two primary constituents: Input Validation and Query Interpretation. The Input Validation module checks whether the user query is in the proper format, free of syntax errors, and adhering to specified constraints. The Query Interpretation module, after validation, interprets the input to extract meaningful intention and corresponding keywords and thus transforms the user's natural language query to a structured format. The cleaned query is then passed to the AI-Powered Layer to produce embeddings and build queries.

C. Embedding Layer

The Embedding Layer is an important component of the system architecture that converts natural language queries to numerical vector representations, which are computationally efficient for semantic retrieval. This Layer uses cutting-edge transformer-based embeddings from Hugging Face and leverages the FAISS index to enable efficient and rapid nearest-neighbor search in high-dimensional vector space.

1) Embeddings: The Embedding Layer is an important component of the system architecture that converts natural language queries to numerical vector representations, which are computationally efficient for semantic retrieval [28]. This Layer uses cutting-edge transformer-based embeddings from Hugging Face and leverages the FAISS index to enable efficient and rapid nearest-neighbor search in high-dimensional vector space [29].

2) Vector Database: To enable effective vector similarity-based retrieval, the framework makes use of FAISS as the underlying indexing mechanism. FAISS is suitable for real-time application-based processing of embedding-based query matching and is specifically designed for the nearest-neighbor search task in high-dimensional space. In order to facilitate effective brute-force nearest-neighbor search using L2 distance measures, the embeddings produced by Hugging Face's transformer model are indexed in an FAISS index that is seeded with IndexFlatL2 [30] FAISS retrieves semantically similar queries by performing a top-k nearest neighbors search in the index given an input query embedding.

D. AI-Powered Layer

From user input, the AI-powered Layer dynamically creates syntactically sound and semantically relevant MongoDB queries. Semantic similarity matching and few-shot prompting with LLMs are the two main submodules that the Layer incorporates.

1) Semantic Similarity Matching: By retrieving contextually similar historical instances from the FAISS index, the Semantic Similarity Matching component aims to improve query accuracy by enabling the language model to function

on contextually similar prior queries. The core of this activity is vector-based retrieval augmented generation (RAG), in which high-confidence retrieved samples are integrated into the prompt to be used in query production [31]. The general steps in this procedure are:

a) User Query Embedding Generation: The Hugging Face transformer-based embedding model is used to encode the incoming query as a dense vector representation.

b) FAISS Similarity Search: The top-N most similar prior queries are returned by comparing the query embedding with the stored vector database.

c) Example Selection and Ranking: High-relevance few-shot examples are chosen as input for downstream LLM-based query generation by using a SemanticSimilarityExampleSelector to rank the returned examples.

2) Few-Shot Prompting: Using a few sample queries and the corresponding MongoDB translations, the Few-Shot Prompting mechanism, which is part of the AI-Powered Layer, improves query generation. The Few-Shot Prompt Selector makes this possible by retrieving the best examples from the user-supplied input using Semantic Similarity Matching. These examples give the LLM Query Generator contextual guidance, ensuring that the generated query follows the format and purpose of previous queries.

E. Database Layer

The Database Layer manages the AI system's generated MongoDB queries and data retrieval. There are two main components to it:

1) MongoDB Connection – This module facilitates smooth communication between the application and the database server by creating and maintaining a secure connection to the store_inventory database. It offers the ability to query various collections, including inventory, sales, discounts, and products.

2) Query Execution Engine – After an AI-Powered Layer-generated MongoDB query is sent to the Query Execution Engine, it is processed and run against the relevant collections. The query is formatted correctly, pertinent data is retrieved, and structured results are returned by the engine. After being retrieved, the data is sent to the Results Processing Layer, where it is formatted and parsed to produce output that is easy to use.

F. Results Processing Layer

The foundation for transforming unformatted database query results into standardised, user-friendly formats is the Results Processing Layer. There are two main parts to it: Raw MongoDB query results from the Database Layer are interpreted by the Query Result Parsing component. It ensures the output is logically related to the user's query, interprets relevant information, and eliminates unnecessary information. The parsed results are converted into an understandable and organised format by User-Friendly Output Formatting. The component ensures that the user can easily interpret the response because it is clear, succinct, and well-structured.

G. Dataset and Experimental Setup

A fictitious dataset that mirrored the structure of an inventory database was used to assess the system. Product_name, product_category, stock_level, discount_percentage, sales_trend, and supplier_details are all included in the 1000 inventory records that make up the dataset.

150 natural language queries that mimicked common inventory management queries—such as stock checks, discount enquiries, sales trends analysis, and multi-condition queries involving filters and aggregations—were hand-picked for query evaluation.

The following parameters were set up in the experimental setup:

- Hardware: Nvidia RTX 4070 GPU, 32 GB RAM, and Intel i7-12700K CPU.
- Programs: Hugging Face's Sentence-Transformers v2.2, FAISS v1.7, MongoDB v6.0, and Python 3.10.
- LLM Model: LLaMA 3.3-70B optimized for MongoDB query structures.
- Synthetic embedding vectors generated from product inventory text data were used in all vector search benchmarking experiments.

IV. CONCEPTUAL FRAMEWORK

The three-tier architecture of the system is depicted in Figure 2: (1) Embedding & Retrieval: user query → transformer embedding → FAISS index → top-k contexts; (2) Prompt Construction: user query + retrieved examples → few-shot prompt; (3) LLM Query Generation: LLaMA processes prompt → MongoDB query → executed by Query Engine. Strong handling of new queries is ensured by this dynamic retrieval-augmented paradigm.

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 21s, 2025
https://theaspd.com/index.php

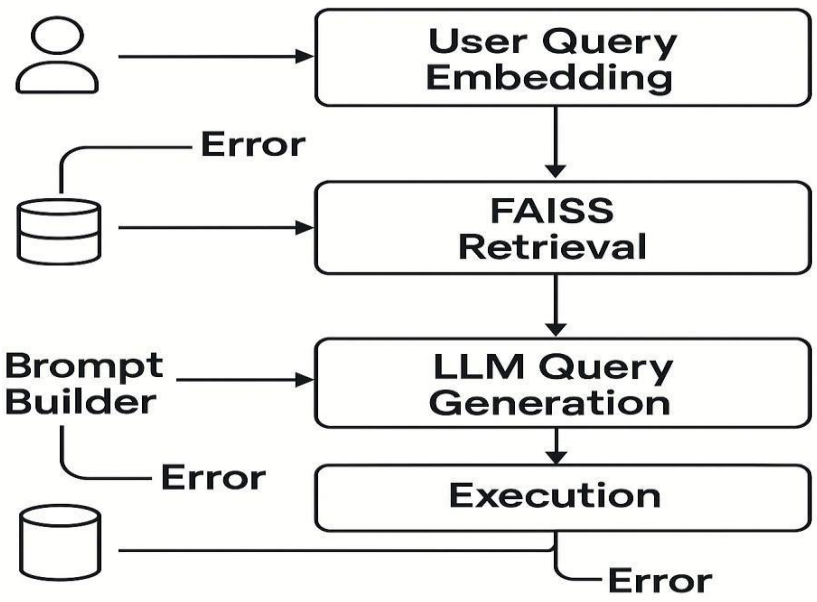

Fig. 2. Conceptual framework for MongoDB retrieval-augmented few-shot query generation.

V. System Architecture

A comprehensive dataflow diagram is shown in Figure 3. The Embedding Service (Python FastAPI) receives the user query from the user interface. The FAISS index is used to query the embedding vector. The Prompt Builder concatenates the retrieved samples. After processing the prompt, LLaMA 3.3-70B outputs a MongoDB query in JSON format. The query is executed on store_inventory by the Query Execution Engine (Node.js service), which then returns the results. Error handling: retry with the top five retrieval results if the generated query fails MongoDB schema validation; if FAISS returns no neighbours (similarity < 0.2), fallback to the zero-shot template. Security: MongoDB's role-based access control and TLS connections.

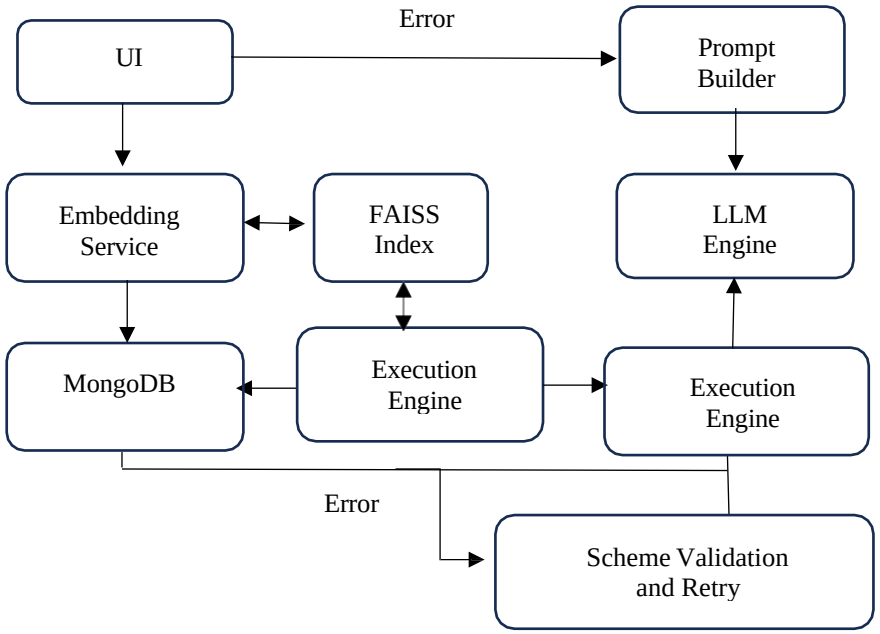

Fig. 3. Detailed system architecture for LLM-driven MongoDB query system

VI. Experiment and Results

The system's capacity to produce precise and executable MongoDB queries from natural language inputs is assessed in the first evaluation stage. Input queries, the system-generated MongoDB query, and the outcome are shown in Fig.

4. The interface gives a clear picture of how the system converts unstructured language inputs into structured database queries, runs those queries, and then reinterprets the results to produce responses that are readable by humans.

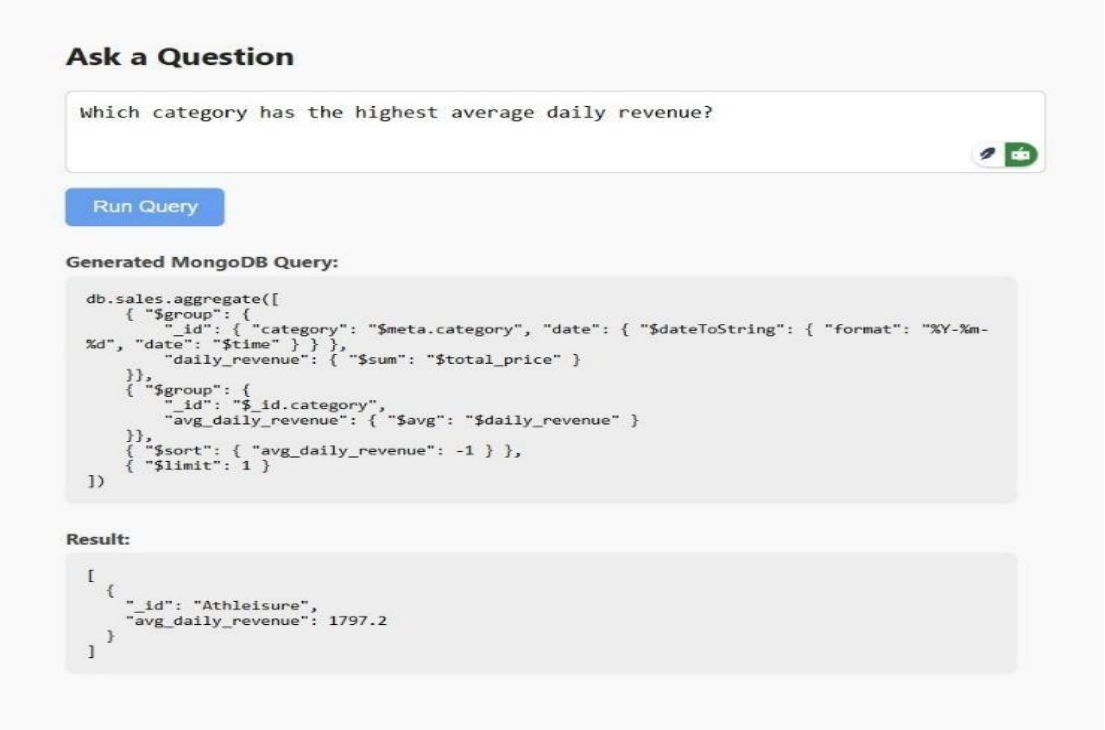

Fig. 4. Web Application for the system

These findings demonstrate excellent logical precision and syntactic accuracy when translating natural language expressions into structured database queries. In order to ensure that database queries appropriately reflect user intent, the generated queries maintain crucial filtering criteria, aggregation functions, and field constraints. Additionally, the final NLP output improves the system's interpretability and usability by reconstructing the returned database results into a comprehensible response.

To systematically assess the accuracy of generated MongoDB queries, a cosine similarity comparison between generated query embeddings and ground-truth anticipated query embeddings is used. This gives an objective, numerical comparison of the structural and semantic similarities between the query and the system's response. The cosine similarity scores of a few selected queries for few-shot, one-shot, and zero-shot learning techniques are displayed in Fig. 5 [32].

With the highest cosine similarity scores—an average of 0.935 on various test queries—the results unequivocally demonstrate that few-shot learning consistently outperforms one-shot and zero-shot learning. The retrieval-augmented example selection mechanism, which dynamically retrieves semantically similar examples from the stored prompt database, is responsible for few-shot learning's superior performance. By reducing the frequency of syntactic errors, misplaced operators, and incorrect nesting of filtering conditions, the retrieval mechanism increases the logical correctness of generated queries.

Conversely, one-shot learning is relatively effective with average similarity scores of 0.90 but performs poorly in scenarios with composite aggregation queries, multi-condition filtering, and nested document structures. The scarcity of multiple contextual instances prevents the model from generalizing across query types, and the synthesized queries exhibit slight structural differences. The performance is as expected with zero-shot learning, with the poorest performance, having an average similarity of 0.826, where some of the queries exhibit drastic deviation from the expected syntax. The inherent cause of the performance drop is the lack of direct reference instances, which leads the model to produce structural inconsistencies and misinterpret the query intent.

A closer inspection of the performance of individual queries indicates that plain filtering queries have imperceptible differences between varying learning paradigms. Nevertheless, complex aggregation and nested condition queries show notable improvements when applying few-shot prompting. This indicates that semantic reinforcement-based few-shot learning is beneficial for structured query generation problems with hierarchical conditions and mathematical aggregations.

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 21s, 2025
https://theaspd.com/index.php

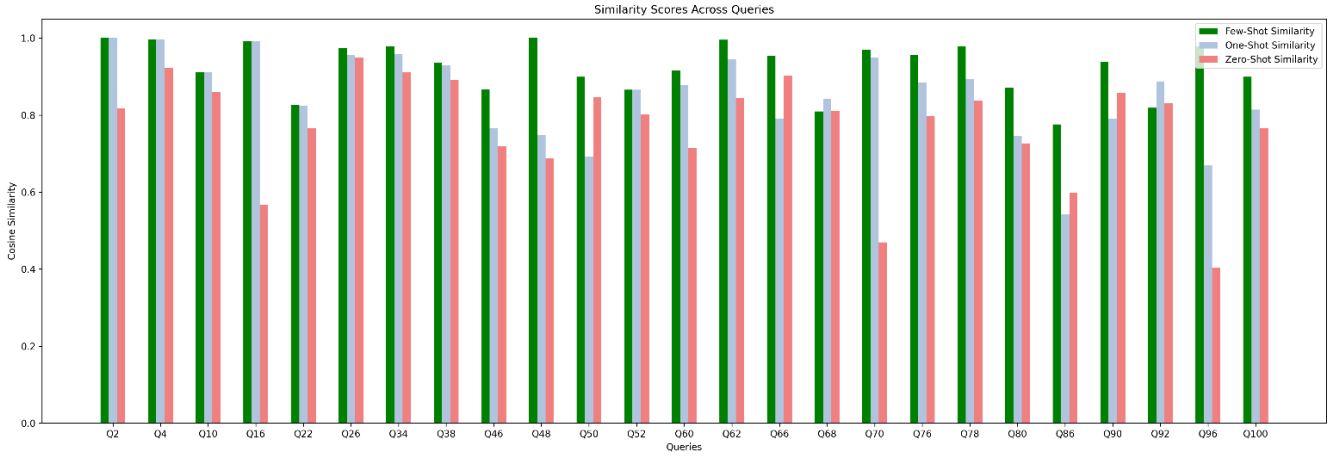

Fig. 5. Cosine Similarity Score Comparison Across Few-Shot, One-Shot, and Zero-Shot Approaches

Aside from query generation accuracy, system latency in finding suitable examples for few-shot prompting is critical to enable real-time execution of queries. Vector retrieval should be optimized for low-latency execution to enable responsiveness in real-world deployment. To evaluate this aspect, we compare index and retrieval performance on three vector search libraries—FAISS, ChromaDB, and Annoy—based on comparison of indexing time and query response time [28]. The experiment is done by building a synthetic dataset of 1000 randomly generated embeddings, the vectorized form of product inventory data.

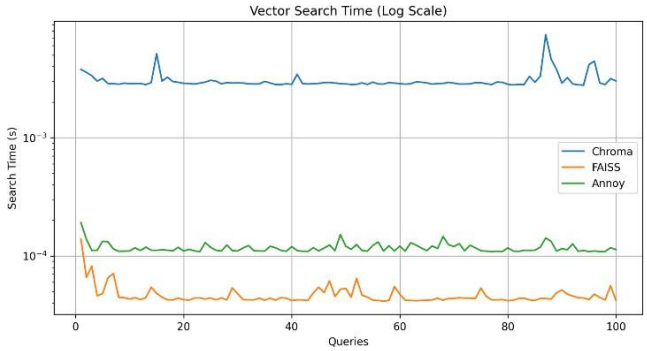

Fig. 6. Query Execution Speed Comparison Across Vector Search Methods

Indexing efficiency is gauged by the computation of the time consumed to construct an index over the data. On the other hand, the retrieval efficiency is gauged by the computation of the average time consumed to execute a top-5 nearest neighbor search for every query embedding. Fig. 6 illustrates a comparative plot of the query execution times of the different frameworks.

FAISS performs better than ChromaDB and Annoy significantly in retrieval latency, with the lowest average execution time for query execution among all the test queries. This is due to FAISS's L2 distance computation optimizations and parallelized brute-force search for nearest neighbor. The experiment shows that FAISS is most appropriate for applications that need real-time retrieval with low-latency query execution as a priority.

Table 2: Performance of Zero-, One-, and Few-Shot Learning

Evaluation Metric	Few-Shot Learning	One-Shot Learning	Zero-Shot Learning
Average Cosine Similarity	0.935	0.900	0.826
Syntax Error Rate (%)	2.5%	5.8%	12.4%
Logical Misinterpretation Rate (%)	3.1%	6.7%	15.9%
Average Query Execution Time (ms)	78 ms	82 ms	85 ms

Table 3: Vector Retrieval Performance by Library.

Vector Search Library	Indexing Time (s)	Average Query Retrieval Time (ms)
FAISS (IndexFlatL2)	1.2 s	42 ms
ChromaDB	1.8 s	65 ms
Annoy	1.4 s	58 ms

Statistical Significance

All experiments were repeated over five random seeds. We report mean ± standard deviation for each metric and conduct paired two-tailed t-tests ($\alpha = 0.05$) comparing few-shot vs. one-shot and few-shot vs. zero-shot. For example, few-shot cosine similarity (0.935 ± 0.008) significantly outperforms one-shot (0.900 ± 0.012), $p < 0.01$.

Table 4: Statistical Significance of Few-Shot vs. One-Shot and Zero-Shot Comparisons.

Comparison	Mean Difference	t-statistic	p-value
Few-shot vs. One-shot	0.035	4.62	< 0.01
Few-shot vs. Zero-shot	0.109	7.15	< 0.001

VII. DISCUSSION

While the system achieves high accuracy in translating natural language queries into executable MongoDB commands, several limitations were observed:

1. Ambiguous Query Interpretations: Queries with vague descriptors (e.g., "best-selling products recently") occasionally lead to generic query structures lacking time-bound constraints.
2. Complex Nested Aggregations: The system struggles with deeply nested aggregations involving multiple collections, often requiring manual prompt engineering to generate accurate queries.
3. Schema Adaptability: The current model assumes a static database schema. Changes to schema structures necessitate retraining or updating the few-shot example repository.
4. Zero-Shot Learning Limitations: As observed, the absence of contextual examples leads to structural inaccuracies, particularly for queries involving composite conditions or mathematical computations.

Ethical Implications: This system processes user queries and inventory data that may be sensitive. All communications are encrypted via TLS 1.3, and access is governed by role-based permissions. To mitigate bias, we monitor query misinterpretation rates across product categories. Automated generation is logged and auditable. Future work includes user-consent dialogues and GDPR-compliant data handling.

VIII. CONCLUSION

This study offers a solid framework for executing queries over MongoDB-based inventory datasets in real-time using natural language. Through the integration of transformer-based embeddings, vector retrieval mechanisms, and LLM-powered query generation, the system efficiently converts unstructured queries into precise and executable MongoDB commands. With a high cosine similarity score of 0.935, the experimental results demonstrate the efficacy of semantic retrieval-augmented few-shot learning, which dramatically improves query accuracy. This method reduces errors and ensures syntactic correctness, especially when handling complex aggregation queries and nested filtering conditions. Because of its parallelised search and optimised L2 distance calculation, FAISS has the lowest query execution latency for real-time retrieval, according to the comparison of vector search engines. FAISS works best in scenarios that call for low-latency semantic similarity measurements because ChromaDB and Annoy have longer retrieval times.

The findings demonstrate that inventory management analytics efficiency can be increased without the need for technical database knowledge by integrating semantic retrieval with LLM-based query construction. This maintains the effectiveness and precision of the query while giving nontechnical users easy access to structured insights with less work. The system is a viable option for intelligent inventory management and decision support systems because of its scalable architecture and optimised retrieval schemes, which ensure real-time performance.

IX. Limitations and Future Work

Limitations:

- For deeply nested multi-collection aggregations, performance deteriorates (the syntax error rate increases to 7%).
- The zero-shot scenario requires more reliable prompt retrieval because it produces a lower semantic match (0.826 cosine similarity).
- It is still necessary to confirm scalability to datasets larger than 10 million records.

Future Work:

- Dynamic Schema Adaptation: use real-time schema reflection to ensure timely grounding.
- Interactive Clarification Dialogue: To clear up user intent, incorporate a two-turn question-answering module.
- Advanced Retrieval Strategies: investigate cross-encoder model re-ranking for zero-shot improvements and hybrid ANN techniques (e.g., HNSW).
- Benchmark Integration: community leaderboard-based open-source assessment of Spider and WikiSQL.

REFERENCES

[1] K. D. Dhole, R. Chandradevan, and E. Agichtein, "An Interactive Query Generation Assistant using LLM-based Prompt Modification and User Feedback," Demo Day at Intelligence Advanced Research Projects Activity â•fi Better Extraction from Text Towards Enhanced Retrieval (IARPA BETTER), vol. 1, Nov. 2023, doi: projectspages/better-demo.html.

[2] Z. Hong, Z. Yuan, H. Chen, Q. Zhang, F. Huang, and X. Huang, "Knowledge-to-SQL: Enhancing SQL Generation with Data Expert LLM," Proceedings of the Annual Meeting of the Association for Computational Linguistics, pp. 10997–11008, Feb. 2024, doi: 10.18653/v1/2024.findings-acl.653.

[3] J. Liu and B. Mozafari, "Query Rewriting via Large Language Models," vol. 14, no. 1, Mar. 2024, Accessed: Aug. 02, 2025. [Online]. Available: https://arxiv.org/pdf/2403.09060

[4] V. Camara, R. Mendonca-Neto, A. Silva, and L. Cordovil, "A Large Language Model approach to SQL-To-Text Generation," Digest of Technical Papers - IEEE International Conference on Consumer Electronics, 2024, doi: 10.1109/ICCE59016.2024.10444148.

[5] A. Kasar, M. M. Tripathi, D. Jadav, T. Chavan, and A. Bathia, "EVOLUTION OF THE PERISHABLE INVENTORY MANAGEMENT LANDSCAPE: A BIBLIOMETRIC ANALYSIS AND FUTURE DIRECTIONS," World Journal of Management and Economics, vol. 18, no. 02, pp. 389–409, Apr. 2025, Accessed: Apr. 22, 2025. [Online]. Available: https://wesro.org/volume-18-issue-02/

[6] S. Koul and A. S. Kasar, "Research Landscape of Sustainable Marketing: Thematic Analysis and Future Trends," Vision, 2024, doi: 10.1177/09722629241231431.

[7] M. Uma, V. Sneha, G. Sneha, J. Bhuvana, and B. Bharathi, "Formation of SQL from natural language query using NLP," ICCIDS 2019 - 2nd International Conference on Computational Intelligence in Data Science, Proceedings, Feb. 2019, doi: 10.1109/ICCIDS.2019.8862080.

[8] E. Mehmood and T. Anees, "Performance Analysis of Not only SQL Semi-Stream Join Using MongoDB for Real-Time Data Warehousing," IEEE Access, vol. 7, pp. 134215–134225, 2019, doi: 10.1109/ACCESS.2019.2941925.

[9] Y. Quan and Z. Liu, "InvAgent: A Large Language Model based Multi-Agent System for Inventory Management in Supply Chains," Jul. 2024, Accessed: Aug. 02, 2025. [Online]. Available: https://arxiv.org/pdf/2407.11384

[10] K. D. Dhole, S. Bajaj, R. Chandradevan, and E. Agichtein, "QueryExplorer: An Interactive Query Generation Assistant for Search and Exploration," Mar. 2024, Accessed: Aug. 02, 2025. [Online]. Available: https://arxiv.org/pdf/2403.15667

[11] A. Kasar, M. M. Tripathi, S. Indolia, A. Rawat, and A. Bathia, "Data-Driven Decision Making for Perishable Food Supply Chains: Insights from Demand Forecasting Models," Advances in Consumer Research, vol. 2, pp. 1461–1468, Feb. 2022, doi: n/a.

[12] X. Liu et al., "A Survey of Text-to-SQL in the Era of LLMs: Where are we, and where are we going?," Aug. 2024, Accessed: Aug. 02, 2025. [Online]. Available: https://arxiv.org/pdf/2408.05109

[13] A. Mohammadjafari, A. S. Maida, R. Gottumukkala, and P. Student, "From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems," Oct. 2024, Accessed: Aug. 02, 2025. [Online]. Available: https://arxiv.org/pdf/2410.01066

[14] Z. Hong et al., "Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL," Jun. 2024, Accessed: Aug. 02, 2025. [Online]. Available: https://arxiv.org/pdf/2406.08426

[15] K. D. Dhole, R. Chandradevan, and E. Agichtein, "Generative Query Reformulation Using Ensemble Prompting, Document Fusion, and Relevance Feedback," May 2024, Accessed: Aug. 02, 2025. [Online]. Available: https://arxiv.org/pdf/2405.17658

[16] T. Falatouri, D. Hrusecka, and T. Fischer, "Harnessing the Power of LLMs for Service Quality Assessment from User-Generated Content," IEEE Access, vol. 12, pp. 99755–99767, 2024, doi: 10.1109/ACCESS.2024.3429290.

[17] B. Li, K. Mellou, B. Zhang, J. Pathuri, and I. Menache, "Large Language Models for Supply Chain Optimization," Jul. 2023, Accessed: Aug. 02, 2025. [Online]. Available: https://arxiv.org/pdf/2307.03875

[18] A. Kasar, M. M. Tripathi, D. Jadav, S. Magar, and A. Bathia, "OPTIMIZATION TECHNIQUES IN PERISHABLE FOOD SUPPLY CHAINS: A SYSTEMATIC LITERATURE REVIEW AND COMPARATIVE ANALYSIS", doi: 10.5281/ZENODO.15470764.

[19] A. S. Korat, "Next-Gen Database Queries: AI-Driven Natural Language Interface for MongoDB," Int J Res Appl Sci Eng Technol, vol. 12, no. 9, pp. 261–269, Sep. 2024, doi: 10.22214/IJRASET.2024.64147.

[20] B. Oza, "Natural Language Query to MongoDB Query".

[21] K. Patel, G. Keshari, D. Powle, S. Ansari, T. Chavan, and A. Khade, "Hybrid NLP Model for Multilingual Sentiment and Emotion Analysis in Poetry," 2024 International Conference on Artificial Intelligence and Quantum Computation-Based Sensor Applications, ICAIQSA 2024 - Proceedings, 2024, doi: 10.1109/ICAIQSA64000.2024.10882405.

[22] T. Yu et al., "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task," Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018, pp. 3911–3921, Sep. 2018, doi: 10.18653/v1/d18-1425.

[23] S. Xue et al., "Demonstration of DB-GPT: Next Generation Data Interaction System Empowered by Large Language Models," Proceedings of the VLDB Endowment, vol. 17, no. 12, pp. 4365–4368, Aug. 2024, doi: 10.14778/3685800.3685876.

[24] T. Yu et al., "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task," Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018, pp. 3911–3921, Sep. 2018, doi: 10.18653/v1/d18-1425.

[25] S. Xue et al., "DB-GPT: Empowering Database Interactions with Private Large Language Models," Dec. 2023, Accessed: Aug. 03, 2025. [Online]. Available: https://arxiv.org/pdf/2312.17449

[26] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference, pp. 3982–3992, Aug. 2019, doi: 10.18653/v1/d19-1410.

[27] Y. Chaudhary, P. Rai, M. Schubert, H. Schütze, and P. Gupta, "Utilizing Language-Image Pretraining for Efficient and Robust Bilingual Word Alignment," Findings of the Association for Computational Linguistics: EMNLP 2022, pp. 154–168, 2022, doi: 10.18653/V1/2022.FINDINGS-EMNLP.12.

[28] E. Mehmood and T. Anees, "Performance Analysis of Not only SQL Semi-Stream Join Using MongoDB for Real-Time Data Warehousing," IEEE Access, vol. 7, pp. 134215–134225, 2019, doi: 10.1109/ACCESS.2019.2941925.

[29] C. Yin and Z. Zhang, "A Study of Sentence Similarity Based on the All-minilm-l6-v2 Model With 'Same Semantics, Different Structure' After Fine Tuning," pp. 677–684, Oct. 2024, doi: 10.2991/978-94-6463-540-9_69.
[30] C. Galli, N. Donos, and E. Calciolari, "Performance of 4 Pre-Trained Sentence Transformer Models in the Semantic Query of a Systematic Review Dataset on Peri-Implantitis," Information 2024, Vol. 15, Page 68, vol. 15, no. 2, p. 68, Jan. 2024, doi: 10.3390/INFO15020068.
[31] L. Wang et al., "Text Embeddings by Weakly-Supervised Contrastive Pre-training," Dec. 2022, Accessed: Aug. 02, 2025. [Online]. Available: https://arxiv.org/pdf/2212.03533
[32] C. Galli, N. Donos, and E. Calciolari, "Performance of 4 Pre-Trained Sentence Transformer Models in the Semantic Query of a Systematic Review Dataset on Peri-Implantitis," Information 2024, Vol. 15, Page 68, vol. 15, no. 2, p. 68, Jan. 2024, doi: 10.3390/INFO15020068.