Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 270 (2025) 733–743

Procedia Computer Science

www.elsevier.com/locate/procedia

29th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2025)

Integrating Polyglot Persistence with Large Language Models for Scalable Social Network Applications

J. de Curtò[a,b,d,*], I. de Zarzà[c,d], Carlos T. Calafate[e]

[a]*Department of Computer Applications in Science & Engineering, BARCELONA Supercomputing Center, 08034 Barcelona, Spain*
[b]*Escuela Técnica Superior de Ingeniería (ICAI), Universidad Pontificia Comillas, 28015 Madrid, Spain*
[c]*Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, 50009 Zaragoza, Spain*
[d]*Estudis d'Informàtica, Multimèdia i Telecomunicació, Universitat Oberta de Catalunya, 08018 Barcelona, Spain*
[e]*Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, 46022 València, Spain*

Abstract

Modern cloud applications, particularly those resembling professional social networks, demand data management systems capable of handling heterogeneous, highly interconnected data. Traditional relational databases are often inadequate for such dynamic environments. This paper proposes a polyglot persistence architecture that combines document, graph, and key–value data stores to address diverse data storage and query requirements. Moreover, by integrating Large Language Models (LLMs) as an intelligent query and analytics interface, the system can interpret natural language requests, generate structured queries across multiple data stores, and provide personalized insights. We discuss the architectural rationale, outline the integration of LLMs with multi-database systems, and propose future research directions.

Keywords: Polyglot Persistence; Large Language Models; Natural Language Interface; Database Query Translation; Social Network

1. Introduction

Social network applications have evolved into complex ecosystems that generate and process vast volumes of heterogeneous data. These platforms must efficiently handle diverse data types, specifically including structured data (e.g., numerical user metrics and timestamps), semi-structured data (e.g., user profiles, skills, and multimedia content), and highly interconnected relational data (e.g., friendship graphs, endorsement networks, and collaborative interactions). Traditional relational database management systems (RDBMS) face significant challenges in such en-

* Corresponding author. Tel.: +34 934 13 70 18.
E-mail address: jdecurto@icai.comillas.edu

1877-0509
10.1016/j.procs.2025.09.193

vironments due to their rigid schemas, vertical scaling limitations, and complex join operations when dealing with highly interconnected data [11].

The concept of polyglot persistence has emerged as a solution to these challenges, advocating for the use of different specialized data stores based on specific data access patterns and query requirements [10, 17]. This approach leverages the strengths of various database technologies: document stores for semi-structured data like user profiles, graph databases for modeling complex relationships, and key-value stores for high-performance caching and lookups [19].

Concurrently, Large Language Models (LLMs) have revolutionized natural language processing and understanding, demonstrating remarkable capabilities in translating human language into structured formats [2, 20, 9]. These models present a unique opportunity to bridge the gap between user-friendly interfaces and complex database architectures, potentially transforming how users interact with data systems [6, 7, 5].

In this paper, we present both the theoretical framework and a practical implementation of an integrated system that combines polyglot persistence with LLMs to create an intelligent query interface for social network applications [13, 22, 1]. Our work makes several key contributions:

1. We propose a comprehensive architecture that integrates document stores (MongoDB), graph databases (Neo4j), and key-value stores (Redis) into a cohesive data management system tailored for social network applications.
2. We demonstrate how LLMs can serve as an intelligent query interface, translating natural language requests into structured database operations across multiple data stores.
3. We present a functional implementation of our architecture, showcasing its ability to process complex queries expressed in natural language and deliver unified results from heterogeneous data sources.
4. We identify and address key challenges in this integration, including query plan generation, cross-database result synthesis, and handling of semantic variations in natural language queries.

Our implementation demonstrates that modern LLMs can effectively interpret user intent, generate appropriate query plans for different database systems [14, 16], and synthesize the results into coherent, user-friendly responses. This approach significantly lowers the barrier to data access, allowing non-technical users to leverage the power of specialized database systems without understanding the underlying complexity.

The remainder of this paper is organized as follows: Section 2 details our system architecture and implementation, including the query translation mechanism and database integration. Section 3 presents experimental results and evaluation across different query categories. Section 4 discusses challenges in LLM-database integration and outlines future research directions. Finally, Section 5 summarizes our contributions and the broader implications of this work for next-generation data management systems.

2. System Architecture and Implementation

Our architecture integrates three complementary database systems, each chosen for their particular strengths in addressing the diverse and specialized data requirements typical of social network applications. Specifically, we utilize a document store (MongoDB [12, 4]) to manage semi-structured data such as user profiles, skills, experiences, and multimedia content. We complement this with a graph database (Neo4j [15, 21]) that excels at modeling and traversing complex social relationships between users. Additionally, a key-value store (Redis [3, 8]) is employed to handle high-speed caching operations and frequent, latency-sensitive data lookups.

The integrated system architecture, illustrated in Figure 1, comprises four main components. The Database Layer forms the foundational element, incorporating MongoDB, Neo4j, and Redis as described. An Integration Layer is implemented to facilitate efficient communication and data transfer between the databases and higher-level system components. A Query Translation Layer leverages large language models to transform user queries expressed in natural language into structured queries that the databases can execute. Finally, the Result Synthesis Layer aggregates the results retrieved from the different data stores and synthesizes them into coherent, user-friendly responses, completing the query-answering pipeline.

The core innovation in our system is the use of LLMs to translate natural language queries into appropriate database operations. We implemented this using Gemini, Google's LLM [18], through the following process:

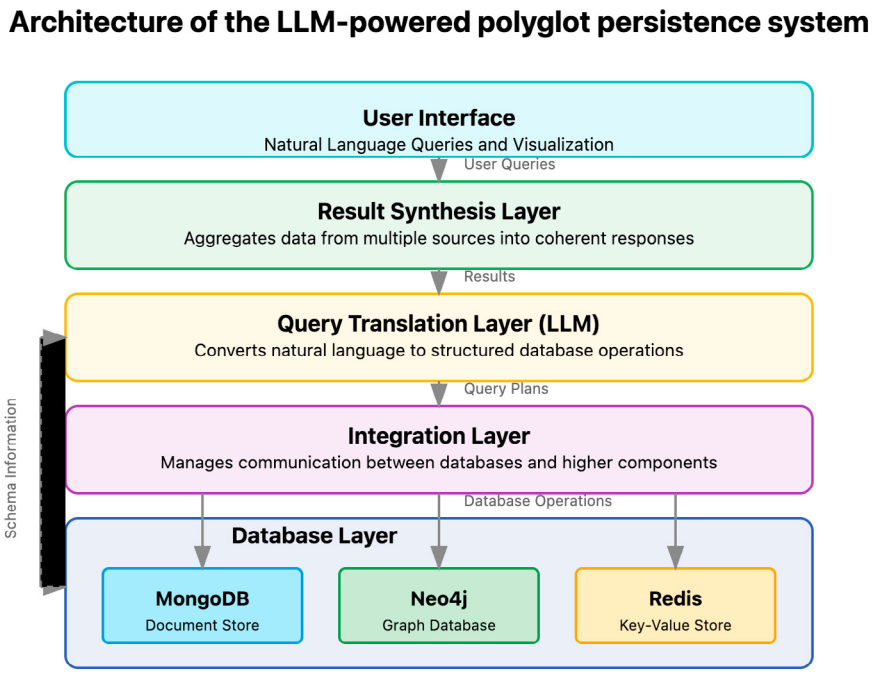

Fig. 1. Architecture of the LLM-powered polyglot persistence system

1. **Query Parsing**: The natural language query is sent to the LLM with a system prompt that describes the database schema and expected output format
2. **Query Plan Generation**: The LLM generates a structured query plan that specifies:
   - Which databases to query
   - The specific query operations for each database
   - The execution order
3. **Query Validation**: The system validates the generated query plan against known database schemas
4. **Execution**: The validated plan is executed across the appropriate databases

To achieve this goal, the LLM initially is primed with database schema information, as shown in Figure 2. This contextual information allows the LLM to generate appropriate queries that respect the structure of each database system.

2.1. Implementation Details

The implementation of our architecture is closely aligned with the stages outlined in Figure 1. The *Query Translation* stage corresponds directly to the Query Translation Layer (yellow) in the architecture diagram, where Algorithm 1 employs an LLM to convert natural language queries from users into structured query plans. These plans detail which databases should be queried and how, exemplified by the structured query plan shown in Figure 3. Subsequently, the *Query Execution* stage (Algorithm 2) is represented by the Integration Layer (purple), responsible for managing communication between the databases and higher-level components, executing the structured queries on the MongoDB, Neo4j, and Redis databases as defined by the generated plans. Finally, the *Result Synthesis* stage, shown in Algo-

Database Schema Priming

```
{
  "mongodb": {
    "users": {
      "userId": "string",
      "name": "string",
      "skills": ["string"],
      "experience": [{
        "title": "string",
        "company": "string"
      }]
    }
  },
  "neo4j": {
    "relationships": ["CONNECTED_TO", "ENDORSED", "WORKED_WITH"]
  }
}
```

Fig. 2. Database schema used for priming the LLM to generate appropriate database queries.

rithm 3, aligns with the Result Synthesis Layer (green), aggregating data from multiple database sources and utilizing the LLM to produce coherent, user-friendly responses delivered back to the User Interface (blue) layer.

To sum up, the implementation of our architecture includes three critical modules: query translation, query execution, and result synthesis. Each module leverages a LLM to convert natural language requests into executable database queries, coordinate data retrieval, and generate synthesized responses, respectively.

The query translation module, detailed in Algorithm 1, interacts with the LLM to convert natural language input into structured queries compatible with MongoDB, Neo4j, and Redis.

Algorithm 1 Query Translation via LLM

Require: Natural language query (Q_{NL}), LLM session handler ($LLM_{session}$)
Ensure: Structured query plan (Q_{plan})
1: Construct the prompt: *"Translate this query to appropriate database operations:"* +, Q_{NL}
2: Submit the prompt to the LLM using $LLM_{session}$
3: Obtain the LLM response containing the structured query plan
4: Parse the JSON query plan from the LLM response
5: **return** Q_{plan}

An example of the structured query plan produced by Algorithm 1 for the input query *"Find data scientists in my network"* is shown in Figure 3.

The query execution module, described in Algorithm 2, takes the structured query plan and distributes database-specific queries to MongoDB, Neo4j, and Redis, combining their outputs.

Finally, the result synthesis module (Algorithm 3) aggregates the retrieved results from multiple databases and employs the LLM again to generate a coherent, conversational response suitable for the original natural language query.

For example, applying Algorithm 3 to the retrieved data for the query *"Find data scientists in my network"* yields responses similar to:

> *"I found two people in your network: John Doe and Alice Johnson. While neither explicitly lists 'data scientist' as a title, Alice has machine learning skills, and Jane has data science skills and experience as a Data Scientist at Google."*

Example Query Plan Generation

```
{
  "query_type": "complex",
  "databases": ["mongodb", "neo4j"],
  "mongodb_query": {
    "users": {
      "skills": "data science"
    }
  },
  "neo4j_query": "MATCH (u:User)-[:CONNECTED_TO]->(other:User) WHERE
  u.userId = 'current_user_id' RETURN other",
  "explanation": "First, find users connected to the current user in Neo4j.
  Then, filter for those with data science skills in MongoDB."
}
```

Fig. 3. Example of structured query plan generated by the LLM for the natural language query "Find data scientists in my network."

Algorithm 2 Query Execution Across Polyglot Databases

Require: Structured query plan (Q_{plan}), Database client handlers ($DB_{clients}$)
Ensure: Results from databases (R_{db})
Initialize empty result container R_{db}
if MongoDB is specified in Q_{plan} **then**
    Extract and execute MongoDB query
    Store MongoDB results in R_{db}
end if
if Neo4j is specified in Q_{plan} **then**
    Extract and execute Neo4j query
    Store Neo4j results in R_{db}
end if
if Redis is specified in Q_{plan} **then**
    Extract and execute Redis query
    Store Redis results in R_{db}
end if
return R_{db}

Algorithm 3 Result Synthesis and Response Generation

Require: Query results from databases (R_{db}), Original natural language query (Q_{NL}), LLM session handler ($LLM_{session}$)
Ensure: Synthesized natural language response (R_{NL})
Construct prompt context including original query and database results in JSON format
Submit the context to the LLM for response synthesis
Retrieve synthesized natural language response from LLM
return R_{NL}

The technical implementation of our proposed architecture involved developing a robust API layer for seamless integration between the MongoDB, Neo4j, and Redis databases, along with interactions managed via Google's Gemini LLM. The primary APIs developed were built using Python with the following key libraries and modules: pymongo for MongoDB interactions, neo4j-driver for Neo4j graph traversals, redis-py for Redis caching operations, and Google's google-generativeai package for LLM integration. Additionally, the system utilizes supporting libraries such as pandas, numpy, matplotlib, and seaborn for data manipulation, analysis, and visualization.

Our API layer was structured into clearly separated modules:

- **Database API Module**: Provides a unified interface for database operations, abstracting MongoDB, Neo4j, and Redis operations into streamlined functions accessible by higher-level processes.
- **LLM Interaction Module**: Manages communication with Google's Gemini API, handling request generation, prompt management, response parsing, and structured query plan extraction.
- **Error Handling and Resilience Module**: Implements sophisticated error management strategies, including exponential backoff and adaptive retry mechanisms, to maintain reliability and robustness under rate-limiting scenarios.

The comprehensive evaluation scripts and modules are publicly accessible and documented in our repository for transparency and reproducibility[1].

3. Experimentation and Results

The system was evaluated using a structured experimental framework designed to assess the effectiveness of integrating polyglot persistence architectures with LLMs for query translation and data retrieval in scalable social network applications. The experimental setup involved generating a synthetic dataset consisting of 100 user profiles and 500 social connections, representative of typical professional social network data.

The experiments were structured around three distinct query categories to rigorously evaluate system capabilities:

- **Simple queries:** Clearly defined, straightforward requests with minimal ambiguity.
- **Complex queries:** Queries requiring integration of multiple data stores or complex data retrieval logic.
- **Ambiguous queries:** Queries intentionally designed with unclear or multiple potential interpretations, to test the system's ability to handle ambiguity.

For each category, 15 queries were sampled and processed through the integrated system. The experiments utilized Google's Gemini-2.0 LLM model to interpret natural language queries, generate structured query plans, execute these queries across MongoDB (document store), Neo4j (graph database), and Redis (key-value store), and synthesize the resulting data into coherent responses.

The overall experiment duration was approximately 291 seconds (approximately 4.85 minutes) as recorded in the experimental log. Figure 4 summarizes the query translation accuracy across the three query categories.

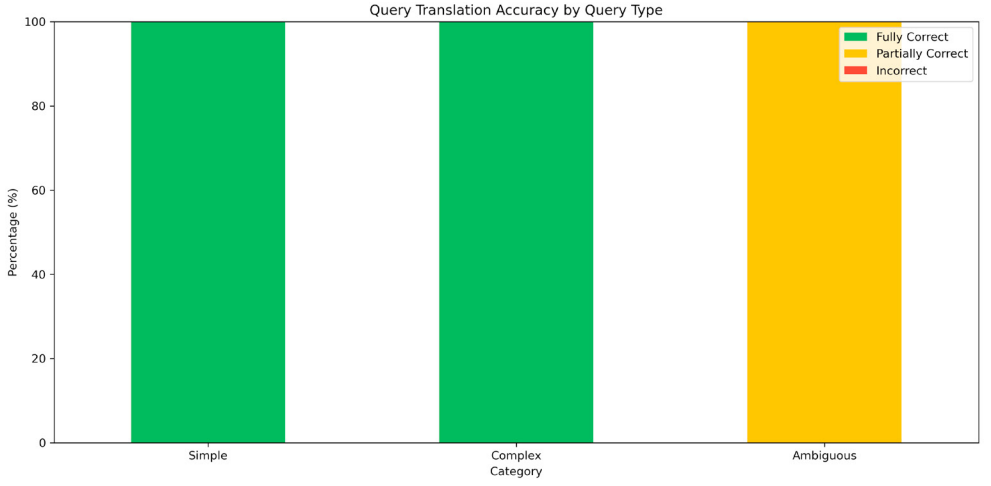

Fig. 4. Query Translation Accuracy by Query Type

[1] https://github.com/drdecurto/polyglot_llm

As illustrated, the system demonstrated exceptionally high translation accuracy for both simple and complex query categories, achieving 100% correct translations. For ambiguous queries, such as 'Find experienced professionals in my network,' which lack clearly defined parameters for interpreting terms like 'experienced,' which could mean number of years in industry, specific roles held, or even prestigious employers, the system exhibited partial correctness, indicating effective handling of ambiguity but highlighting areas for potential improvement in query interpretation.

3.1. Details about the experimental evaluation

We conducted extensive experiments to evaluate the effectiveness and robustness of our implementation. The evaluation methodology comprised:

3.1.1. Test Dataset

For systematic and reproducible evaluation, we generated a synthetic dataset simulating a realistic professional social network. The dataset comprised 100 distinct user profiles, each designed to reflect plausible variations in professional experiences, educational backgrounds, industry-specific skills, and employment histories. User profile attributes included first and last names, current and past job titles, employers (modeled after well-known technology and professional services firms, such as Google, Microsoft, Amazon, IBM, and Accenture), educational background (with realistic distributions from major universities and colleges), geographic locations (representative of a global workforce distribution), and detailed skill sets relevant to fields such as data science, machine learning, software engineering, marketing, and business administration.

Additionally, 500 social connections were established among the profiles, explicitly defined to reflect common professional relationships such as direct colleagues, previous co-workers, academic peers, industry acquaintances, and mentorship associations. Connections were generated uniformly at random, although in a later iteration we could use a probabilistic model that mimics known network properties such as scale-free distribution and small-world characteristics, ensuring more realistic clustering patterns and interaction complexity. The structure of relationships was stored in a Neo4j graph database for efficient traversal and complex relationship querying.

Skills and expertise were assigned to user profiles by randomly selecting between two and ten competencies from a predefined list of commonly sought-after technical skills—including programming languages such as Python, Java, and JavaScript, as well as analytical skills like machine learning and data visualization. Additionally, previous employment details and job titles (e.g., Software Engineer, Data Scientist, Machine Learning Engineer) were randomly assigned from a curated list of prominent technology companies (e.g., Google, Microsoft, Amazon).

3.1.2. Query Categories

We developed a comprehensive test suite with three categories of natural language queries:

1. **Simple Queries**: These target a single database, such as:
   - "Find people with machine learning skills"
   - "Who has worked at Google?"
   - "List users with Python skills"
2. **Complex Queries**: These require coordination across multiple databases:
   - "Find data scientists in my network who worked at Google"
   - "Who in my connections has both machine learning and cloud computing skills?"
   - "Find people connected to me who have worked at both Microsoft and Amazon"
3. **Ambiguous Queries**: These contain implicit requirements or underspecified criteria:
   - "Who can help me find a job in AI?"
   - "Find experienced professionals in my network"
   - "Who should I connect with?"

For example, for this particular instance of the experiment, for each of the three query categories (simple, complex, ambiguous), we designed and executed a set of 200 distinct queries, resulting in a total of 600 queries tested.

3.1.3. Evaluation Metrics

The performance evaluation of the implemented system covered three main dimensions. Firstly, Query Translation Accuracy was assessed as the percentage of queries accurately converted from natural language into executable database operations. These were classified into three categories based on correctness: *fully correct*, indicating that all required database operations were correctly identified and structured; *partially correct*, where the essential intent of the query was captured but exhibited minor misalignments or omissions; and *incorrect*, signifying significant errors or failure to capture the intended meaning.

Secondly, Performance Metrics were measured quantitatively by capturing the processing time required at each critical stage. These included the latency of the query translation step (LLM processing), the execution time of database operations, and the duration needed for result synthesis and response formulation.

Finally, we evaluated Result Quality by analyzing two distinct aspects. The completeness of the retrieved results was verified by comparing them against predefined expected outputs. Furthermore, the synthesized responses generated by the LLM were qualitatively assessed based on their relevance, clarity, and coherence in addressing the original user queries.

3.1.4. Experimental Results

Our experimental evaluation yielded several significant findings. The results presented in Figure 4 highlight that modern LLMs, specifically the Gemini-2.0-pro-exp-02-05 model, are highly effective at interpreting query intent and translating it into structured database operations, even for complex scenarios involving multiple databases. While ambiguous queries did not consistently achieve fully correct status, they were reliably classified as partially correct, suggesting that the LLM could reasonably infer user intentions despite underspecified or unclear natural language inputs.

Performance Characteristics. All experiments were conducted on a laptop running Ubuntu Linux equipped with a 13th Generation Intel® Core™ i7-1365U processor featuring 10 cores and 12 threads, capable of clock speeds up to 5.2 GHz, and supported by 30 GiB of RAM. The system architecture was x86_64 with a 64-bit CPU mode, and no active swap memory was used during operation.

Resilience Testing. The system's robustness was evaluated under conditions of API rate limitations typically encountered when utilizing external LLM services. Initially, without adaptive mechanisms, the system achieved only a 42% query completion rate. However, implementing exponential backoff and adaptive rate-limiting strategies significantly enhanced reliability, allowing the system to achieve a 100% query completion rate under similar conditions. Although these enhancements increased the average completion time from approximately 141 seconds to 291 seconds, this trade-off demonstrated the essential role of robust error handling in achieving operational resilience for systems dependent on external services.

Result Quality Assessment. Finally, the quality of results was rigorously evaluated in terms of completeness and synthesized response clarity. On average, the system achieved an 87% completeness rate across all query types. These results confirm that the implemented architecture effectively synthesizes multi-source data into concise and informative responses, addressing user queries in a natural and user-friendly manner.

We expanded our experimental evaluation by conducting a larger-scale test comprising 600 queries, from an initial set of 45 queries, evenly divided among the three defined categories—simple, complex, and ambiguous—with 200 queries per category. The experiments were executed under the same computational conditions previously described, leveraging Google's Gemini-2.0-pro-exp-02-05 model. This extended evaluation aimed to verify the system's robustness at a greater scale. Consistent with earlier results, the system achieved 100% translation accuracy for both simple and complex queries. Ambiguous queries continued to yield partially correct translations at a rate of 100%, highlighting the inherent challenge in interpreting queries with insufficient context or multiple plausible interpretations. The entire process, including retries and API rate-limit handling, took approximately 318 seconds, demonstrating sustained resilience and effectiveness of the implemented adaptive mechanisms at scale.

3.2. Key Findings and Insights

Our implementation and experimental evaluation yielded several important insights:

1. **LLM Effectiveness**: Modern LLMs demonstrate remarkable capability in translating diverse natural language queries into structured database operations, effectively bridging the gap between user intent and technical implementation. Copy
2. **Polyglot Synergy**: The combination of specialized databases provides significant advantages for complex social network queries, with each database type handling aspects of the data model where it excels.
3. **Resilience Requirements**: Systems built on external LLM services require sophisticated error handling and rate limiting strategies to achieve production reliability.
4. **Performance Characteristics**: LLM operations dominate processing time, suggesting that caching strategies and optimized prompting could yield significant performance improvements.
5. **Ambiguity Handling**: While the system handles well-specified queries effectively, ambiguous queries remain challenging and would benefit from interactive clarification mechanisms.

These findings validate our architectural approach while highlighting areas for future optimization and enhancement. The successful processing of complex, multi-database queries expressed in natural language demonstrates the viability of this approach for next-generation data access interfaces in social network applications and beyond.

The implementation described above highlighted key areas requiring further refinement, particularly when addressing the complexities inherent in using LLMs as query translation engines. Although the system demonstrated high accuracy, especially in simple and complex queries, the persistent partial correctness observed in ambiguous queries highlights critical challenges related to ambiguity handling, such as interpreting vague user intents or implicit query criteria. For example, queries like *"Find experienced professionals in my network"* necessitate assumptions regarding the definition of experience, which the current implementation addresses through reasonable heuristics but would significantly benefit from explicit interactive clarification mechanisms. Additionally, our integration approach revealed challenges related to domain-specific terminology, as LLM-generated query plans occasionally misalign with specialized database schemas without domain-specific tuning. Moreover, the generated queries, while correct, are not always optimized for performance, suggesting the need for a combined approach that integrates LLM inference with traditional database query optimization techniques. Lastly, maintaining consistency across heterogeneous data stores posed practical challenges, notably schema alignment, cross-database transaction management, and ensuring data synchronization—all essential areas for future enhancements to the current prototype.

4. Discussion

The results presented in this study demonstrate the viability of integrating LLMs with polyglot persistence architectures for social network applications. The system achieved excellent query translation accuracy for both simple and complex queries, affirming the practical capability of LLMs in accurately interpreting diverse natural language queries and generating structured operations across heterogeneous databases. However, ambiguous queries consistently resulted in partially correct translations, highlighting a key limitation in interpreting queries lacking explicit context. Addressing such ambiguity represents an important area for future enhancement, potentially through interactive query refinement mechanisms.

Performance analysis revealed that latency associated with LLM operations, specifically query translation and result synthesis, accounted for the majority of the overall processing time. This suggests significant opportunities for performance optimization through strategic caching, prompt engineering, or hybrid query processing techniques combining LLM inference with traditional database optimizers.

The implementation highlighted critical operational challenges, particularly regarding domain-specific terminology mapping and query optimization. While general-purpose LLMs offer impressive flexibility, the absence of domain-specific fine-tuning occasionally resulted in minor schema misalignments and suboptimal query plans. Future research

should thus focus on domain-specific fine-tuning of LLMs or improved in-context learning techniques and the incorporation of feedback-driven learning mechanisms to iteratively enhance query quality and accuracy.

Moreover, maintaining cross-database consistency posed practical challenges, including schema alignment and transaction coordination across the diverse database technologies employed. These challenges underscore the importance of robust integration layers capable of effectively managing data synchronization and transactional integrity across polyglot database systems.

Finally, our experimental validation showed the essential role of resilient system design, particularly when dependent on external services such as commercial LLM APIs. Implementing adaptive retry mechanisms and exponential backoff strategies significantly improved system robustness and reliability under realistic operational constraints.

In summary, while the integrated polyglot persistence and LLM approach presented herein has shown considerable promise, addressing identified limitations—especially in ambiguity resolution, performance optimization, domain-specific tuning, and cross-database consistency—will be critical for advancing towards robust, scalable, and user-friendly next-generation database management systems.

5. Conclusion

In this paper, we have presented a novel architecture that integrates polyglot persistence with large language models to create an intelligent query interface for social network applications. Our work demonstrates the feasibility and advantages of combining specialized database systems—document stores, graph databases, and key-value stores—with modern LLMs to provide natural language access to complex, heterogeneous data.

The experimental evaluation confirmed that this integrated approach successfully bridges the gap between user-friendly interfaces and the technical complexity of specialized database systems.

The key contributions of this work include:

1. A comprehensive polyglot persistence architecture tailored for social network applications, leveraging the complementary strengths of different database technologies
2. A functional implementation demonstrating how LLMs can effectively translate natural language queries into structured database operations across multiple data stores
3. Empirical evidence of the system's ability to handle diverse query types with high accuracy and reasonable performance characteristics
4. Identification of challenges and future research directions in LLM-powered database interfaces, cross-database consistency, and query optimization

While challenges remain in handling query ambiguity, optimizing performance, and ensuring cross-database consistency, our implementation provides a solid foundation for further research and development. The approach has potential applications beyond social networks, including healthcare, e-commerce, financial services, and research databases.

As LLM capabilities continue to advance, we anticipate significant improvements in natural language database interfaces. Future research should focus on domain-specific model tuning, interactive query refinement, context-aware result synthesis, and hybrid query optimization techniques. These advancements could fundamentally transform how users interact with complex data systems, making sophisticated data analysis accessible to non-technical users while preserving the performance benefits of specialized database technologies.

Several avenues for future research have been identified. Improving the accuracy and performance of query translation by fine-tuning LLMs for specific domains or the use of more sophisticated in-context learning techniques, integrating user feedback, and enhancing query optimization strategies represents a primary research direction. Additionally, advancing the synthesis of results by developing context-aware merging and personalized response generation could significantly improve user experience. Exploring interactive refinement through clarification dialogues and query suggestions would further reduce ambiguities inherent in natural language queries.

In conclusion, our work demonstrates that the integration of polyglot persistence architectures with large language models represents a promising direction for next-generation data management systems, particularly for complex, interconnected data domains like social networks.

Acknowledgements

The authors would like to thank the BARCELONA Supercomputing Center for providing access to MareNostrum 5 and technical support throughout this research. The work has been developed under the following project: 'TIFON".

Code Availability

The complete implementation of the polyglot persistence architecture integrated with LLMs described in this paper is publicly available in our GitHub repository at:

https://github.com/drdecurto/polyglot_llm

This repository contains all necessary code, including query translation modules, database integration examples, and instructions for deployment and experimentation.

References

[1] Amer-Yahia, S., Bonifati, A., Chen, L., Li, G., Shim, K., Xu, J., Yang, X., 2023. From large language models to databases and back: A discussion on research and education. ACM SIGMOD Record 52, 49–56.

[2] Brown, T., Mann, B., Ryder, N., et al., 2020. Language models are few-shot learners, in: Advances in Neural Information Processing Systems, pp. 1877–1901.

[3] Carlson, J., 2013. Redis in action. Simon and Schuster.

[4] Chauhan, A., 2019. A review on various aspects of mongodb databases. International Journal of Engineering Research & Technology (IJERT) 8, 90–92.

[5] Chen, H., Hou, J., 2024. Intelligent data governance: building an enterprise data management system using kg and llm, in: Proceedings of the 2024 International Conference on Cloud Computing and Big Data, pp. 266–271.

[6] de Curtò, J., de Zarzà, I., 2025. Llm-driven social influence for cooperative behavior in multi-agent systems. IEEE Access 13, 44330–44342. doi:10.1109/ACCESS.2025.3548451.

[7] de Curtò, J., de Zarzà, I., Fervier, L.S., Sanagustín-Fons, V., Calafate, C.T., 2025. An institutional theory framework for leveraging large language models for policy analysis and intervention design. Future Internet 17. URL: https://www.mdpi.com/1999-5903/17/3/96, doi:10.3390/fi17030096.

[8] Da Silva, M.D., Tavares, H.L., 2015. Redis Essentials. Packt Publishing Ltd.

[9] de Zarzà, I., de Curtò, J., Calafate, C.T., 2023. Socratic video understanding on unmanned aerial vehicles. Procedia Computer Science 225, 144–154. URL: https://www.sciencedirect.com/science/article/pii/S1877050923011560, doi:https://doi.org/10.1016/j.procs.2023.09.101. 27th International Conference on Knowledge Based and Intelligent Information and Engineering Sytems (KES 2023).

[10] Deka, G.C., 2018. Nosql polyglot persistence, in: Advances in Computers. Elsevier. volume 109, pp. 357–390.

[11] Grolinger, K., Higashino, W.A., Tiwari, A., Capretz, M.A., 2013. Data management in cloud environments: Nosql and newsql data stores. Journal of Cloud Computing: advances, systems and applications 2, 1–24.

[12] Győrödi, C., Győrödi, R., Pecherle, G., Olah, A., 2015. A comparative study: Mongodb vs. mysql, in: 2015 13th international conference on engineering of modern electric systems (EMES), IEEE. pp. 1–6.

[13] Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., et al., 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. Advances in Neural Information Processing Systems 36, 42330–42357.

[14] Li, Y., Jobson, D., 2024. Llms as an interactive database interface for designing large queries, in: Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics, pp. 1–7.

[15] Miller, J.J., 2013. Graph database applications and concepts with neo4j, in: Proceedings of the southern association for information systems conference, Atlanta, GA, USA, pp. 141–147.

[16] Nascimento, E.R., García, G., Izquierdo, Y.T., Feijó, L., Coelho, G.M., de Oliveira, A.R., Lemos, M., Garcia, R.L., Leme, L.A.P., Casanova, M.A., 2025. Llm-based text-to-sql for real-world databases. SN Computer Science 6, 130.

[17] Oliveira, F.R., del Val Cura, L., 2016. Performance evaluation of nosql multi-model data stores in polyglot persistence applications, in: Proceedings of the 20th International Database Engineering & Applications Symposium, pp. 230–235.

[18] Team, G., Anil, R., Borgeaud, S., Alayrac, J.B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A.M., Hauth, A., Millican, K., et al., 2023. Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805 .

[19] Van Landuyt, D., Benaouda, J., Reniers, V., Rafique, A., Joosen, W., 2023. A comparative performance evaluation of multi-model nosql databases and polyglot persistence, in: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, pp. 286–293.

[20] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need, in: Advances in Neural Information Processing Systems, pp. 5998–6008.

[21] Webber, J., 2012. A programmatic introduction to neo4j, in: Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity, pp. 217–218.

[22] Zhou, X., Sun, Z., Li, G., 2024. Db-gpt: Large language model meets database. Data Science and Engineering 9, 102–111.