Polyglot Persistence with Large Language Models

Journal Title
XX(X):1–12

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

J. de Curtò[1,2,4], I. de Zarzà[3,4] and Carlos T. Calafate[5]

Abstract

Modern data-intensive applications demand intelligent systems capable of managing heterogeneous, highly interconnected data across multiple specialized storage backends. Yet accessing such systems typically requires expertise in multiple query languages, e.g. SQL, Cypher, MongoDB syntax, limiting accessibility for non-technical users. This paper presents a comprehensive architecture that integrates polyglot persistence, combining document stores, graph databases, key-value caches, and relational data warehouses, with Large Language Models (LLMs) to provide natural language query interfaces. Our implementation compares two Google Gemini model variants,Gemini 3 Pro and Gemini 3 Flash, for translating natural language queries into structured operations across PostgreSQL data warehouses, MongoDB document stores, Neo4j graph databases, and Redis caches. Experimental evaluation across 39 queries spanning six categories reveals a clear accuracy-latency trade-off: Gemini 3 Pro achieves 82.1% fully correct translations with average latency of 12.9–26.5 seconds, while Gemini 3 Flash achieves 76.9% accuracy but with significantly reduced latency of 9.0–11.5 seconds (approximately 1.76× faster). Both models achieve 100% combined accuracy (correct plus partial) with zero incorrect translations. Cross-domain validation comparing Traffic/BI (warehouse-centric) with Social Network (graph-centric) applications demonstrates that translation accuracy improves from 60.0% to 82.1% when moving to structured dimensional schemas, while the architecture adapts effectively across fundamentally different workload patterns. Performance analysis reveals that LLM translation time dominates overall latency (>99%), while database execution remains negligible (<55ms), highlighting opportunities for optimization through caching and prompt engineering. This work contributes a generalizable framework for LLM-powered polyglot persistence systems, comprehensive evaluation methodology for natural language database interfaces, and empirical insights into model selection and domain adaptation trade-offs.

Keywords

Polyglot Persistence, Large Language Models, Natural Language Interface, Data Warehouse, Query Translation, Business Intelligence, Semantic Web

1 Introduction

Modern cloud applications increasingly require sophisticated data management systems capable of handling diverse data types, complex relationships, and multiple access patterns. Traditional relational database management systems (RDBMS), while robust for structured data and transactional workloads, face significant challenges when confronted with the heterogeneous data landscapes typical of contemporary applications (Grolinger et al. 2013). These challenges manifest in rigid schema requirements, vertical scaling limitations, and performance degradation when executing complex join operations across highly interconnected datasets.

The concept of polyglot persistence has emerged as a compelling solution to these challenges, advocating for the strategic deployment of different specialized data stores based on specific data characteristics and query requirements (Deka 2018; Oliveira and del Val Cura 2016). This approach leverages the complementary strengths of various database technologies: document stores excel at managing semi-structured data with flexible schemas, graph databases provide efficient traversal of complex relationships, key-value stores deliver high-performance caching and low-latency lookups, and relational data warehouses support sophisticated analytical queries with dimensional modeling (Van Landuyt et al. 2023).

Concurrently, Large Language Models (LLMs) have demonstrated remarkable capabilities in natural language understanding and generation (Brown et al. 2020; Vaswani et al. 2017), presenting unique opportunities to bridge the gap between user-friendly interfaces and the technical complexity inherent in polyglot persistence architectures (de Curtò and de Zarzà 2025). By enabling users to express

[1]Department of Computer Applications in Science & Engineering, BARCELONA Supercomputing Center, 08034 Barcelona, Spain
[2]Escuela Técnica Superior de Ingeniería (ICAI), Universidad Pontificia Comillas, 28015 Madrid, Spain
[3]Human Centered AI, Data & Software, LUXEMBOURG Institute of Science and Technology, L-4362 Esch-sur-Alzette, Luxembourg
[4]Estudis d'Informàtica, Multimèdia i Telecomunicació, Universitat Oberta de Catalunya, 08018 Barcelona, Spain
[5]Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, 46022 València, Spain

Corresponding author:
J. de Curtò, Department of Computer Applications in Science & Engineering, BARCELONA Supercomputing Center, Plaça Eusebi-Güell 1–3, 08034 Barcelona, Spain.
Email: jdecurto@icai.comillas.edu

queries in natural language rather than specialized query languages (SQL, Cypher, MongoDB query syntax), LLMs can democratize access to sophisticated data systems (Li et al. 2023; Zhou et al. 2024; Amer-Yahia et al. 2023).

de Curtò et al. (2025) introduced a polyglot persistence architecture integrated with LLMs for social network applications, demonstrating the feasibility of natural language query translation across heterogeneous database systems. This paper contributes in the following:

1. Generalizing the architecture to support business intelligence applications, specifically urban traffic analytics.
2. Implementing a comprehensive data warehouse schema using dimensional modeling (star schema) with PostgreSQL.
3. Developing an extensive evaluation framework spanning six query categories and 39 test queries.
4. Comparing two LLM variants (Gemini 3 Pro and Gemini 3 Flash), representing high-accuracy and low-latency configurations within the same model family, to characterize the accuracy-latency trade-off.
5. Providing cross-domain validation comparing Social Network and Traffic/BI applications.
6. Identifying category-specific challenges and opportunities for LLM-powered database interfaces.

The remainder of this paper is organized as follows: Section 2 reviews related work in polyglot persistence, natural language interfaces for databases, and LLM applications in data management. Section 3 presents the system architecture, including the data warehouse schema and query translation pipeline. Section 4 describes the experimental methodology, including dataset characteristics, query categories, and evaluation metrics. Section 5 presents comprehensive experimental results across translation accuracy, performance, and result quality dimensions, including a detailed model comparison. Section 6 provides cross-domain validation comparing with social networks. Section 7 discusses the implications of our findings, identifies limitations, and outlines future research directions. Finally, Section 8 summarizes our contributions.

2 Related Work

The term "polyglot persistence" was popularized to describe the practice of using multiple data storage technologies within a single application, selected based on the nature of the data and access patterns (Deka 2018). This approach acknowledges that no single database technology optimally serves all data management requirements. Grolinger et al. (2013) provide a comprehensive survey of NoSQL and NewSQL data stores in cloud environments, establishing the foundation for understanding when different storage paradigms are appropriate. Oliveira and del Val Cura (2016) present performance evaluations of NoSQL multi-model databases in polyglot persistence applications, demonstrating significant performance variations across different workload patterns. More recently, Van Landuyt et al. (2023) conducted comparative performance evaluations highlighting the trade-offs between multi-model NoSQL databases and traditional polyglot persistence approaches.

The challenge of translating natural language queries into structured database operations has been studied extensively. Li et al. (2023) introduced a comprehensive benchmark for large-scale database grounded text-to-SQL, demonstrating both the capabilities and limitations of current approaches. Nascimento et al. (2025) provide an extensive analysis of LLM-based text-to-SQL systems for real-world databases, identifying key challenges including schema complexity, ambiguity resolution, and domain-specific terminology. Li and Jobson (2024) explore LLMs as interactive database interfaces for designing complex queries, showing promise for iterative query refinement.

The emergence of large language models such as GPT-4, Claude, and Gemini (Team et al. 2023) has opened new possibilities for natural language database interfaces. Zhou et al. (2024) present DB-GPT, demonstrating how large language models can be integrated with database systems to provide intelligent query interfaces. Amer-Yahia et al. (2023) discuss the bidirectional relationship between LLMs and databases, highlighting both opportunities and challenges for research and education. Chen and Hou (2024) explore intelligent data governance using knowledge graphs and LLMs, showing how these technologies can complement each other in enterprise data management systems.

The Semantic Web vision of machine-readable, interoperable data shares fundamental goals with polyglot persistence systems, both seek to enable intelligent query processing across heterogeneous data sources. Knowledge graphs, built on the Resource Description Framework (RDF) and queried via SPARQL (SPARQL Protocol and RDF Query Language), represent a fifth paradigm complementing the four database technologies in our architecture (Hogan et al. 2021). Recent work has explored LLM-based SPARQL generation (Yang et al. 2023), demonstrating similar translation challenges to those we observe with SQL and Cypher. Our dimensional modeling approach, while not employing RDF directly, follows semantic modeling principles: dimension tables encode domain ontologies (vehicle types, contributing factors), and the star schema's explicit relationships parallel RDF's subject-predicate-object triples.

3 System Architecture

Our architecture integrates four complementary database systems within a unified query processing framework, orchestrated by an LLM-powered translation layer. Figure 1 illustrates the high-level system design.

The architecture follows a layered design pattern with clear separation of concerns: (1) the *Interface Layer* accepts natural language queries from users, (2) the *Translation Layer* leverages LLMs to generate database-specific query plans, (3) the *Routing Layer* dispatches queries to appropriate backends based on the generated plan, (4) the *Execution Layer* manages concurrent query execution across heterogeneous systems, and (5) the *Synthesis Layer* aggregates results into coherent natural language responses. This modular design enables independent scaling of components and facilitates the addition of new database backends without architectural changes.

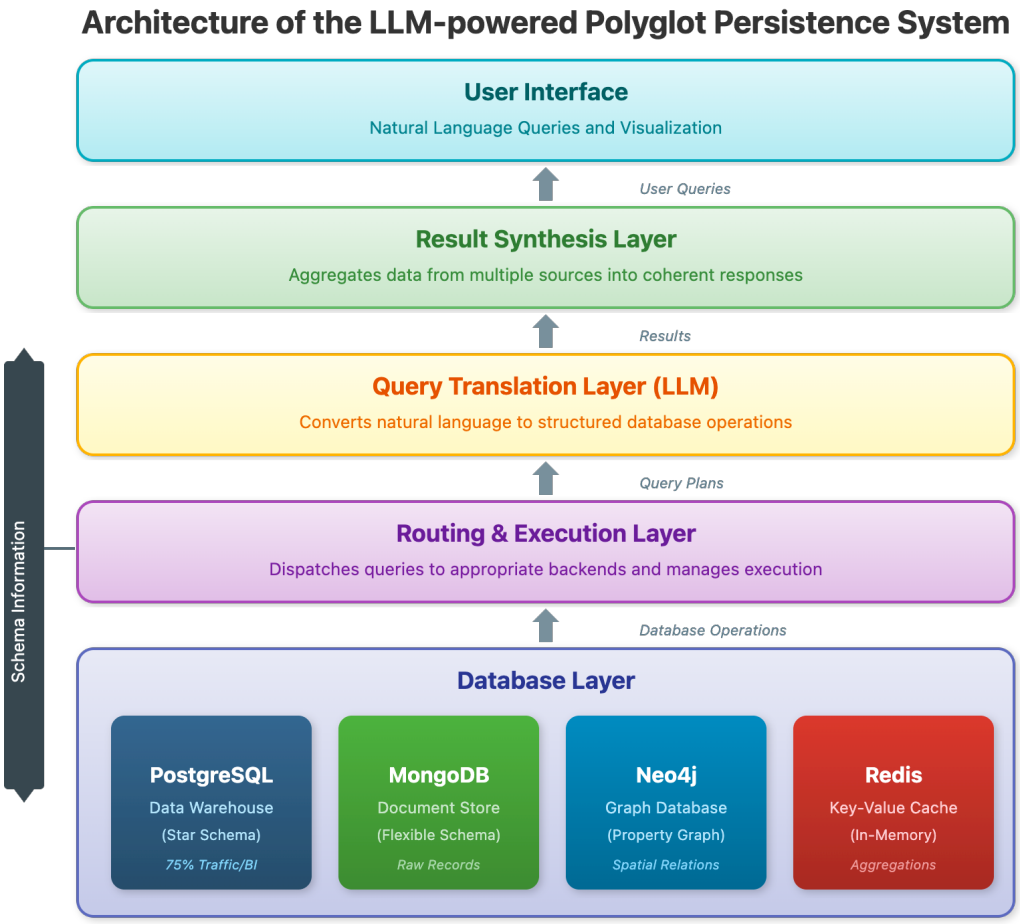

Figure 1. Architecture of the LLM-powered polyglot persistence system showing the five-layer design with four heterogeneous database backends. PostgreSQL serves as the primary analytical store (75% of Traffic/BI queries).

3.1 Database Layer

The database layer comprises four specialized systems, each selected for its strengths in handling specific data patterns and query types:

PostgreSQL Data Warehouse: Implements a dimensional model (star schema) optimized for analytical queries. The schema includes fact tables for collisions and traffic volumes, with dimension tables for location, date, time, vehicle type, and contributing factors. Bridge tables handle many-to-many relationships between facts and dimensions. PostgreSQL was selected for its robust support of complex analytical functions including window functions, common table expressions (CTEs), and statistical aggregations (CORR, STDDEV, VARIANCE).

MongoDB Document Store (Győrödi et al. 2015; Chauhan 2019): Stores semi-structured collision records and traffic volume measurements with flexible schemas, enabling storage of varying attribute sets and nested documents. MongoDB's aggregation pipeline supports complex transformations while maintaining schema flexibility for evolving data requirements.

Neo4j Graph Database (Miller 2013; Webber 2012): Models spatial relationships between locations, boroughs, and traffic segments, enabling efficient graph traversals for proximity and connectivity queries. The property graph model represents locations as nodes with edges encoding spatial adjacency and traffic flow relationships.

Redis Cache (Carlson 2013; Da Silva and Tavares 2015): Provides high-speed caching for frequently accessed aggregations and query results, reducing load on primary data stores. Redis sorted sets maintain pre-computed rankings (e.g., top collision locations), while hash structures store cached aggregation results with configurable TTL.

3.2 Data Warehouse Schema

The NYC Traffic data warehouse implements a comprehensive star schema optimized for analytical query patterns. Table 1 presents the complete schema structure.

The schema incorporates two bridge tables to handle many-to-many relationships: `BRIDGE_COLLISION_VEHICLE` links collisions to multiple involved vehicle types, and `BRIDGE_COLLISION_FACTOR` links collisions to multiple contributing factors.

3.3 Query Translation Pipeline

The query translation process, powered by Google's Gemini models (Team et al. 2023), proceeds through four stages as formalized in Algorithm 1.

The algorithm operates in four phases: (1) query classification determines the semantic type and target databases, (2) query plan generation produces database-specific queries, (3) validation ensures schema compliance with iterative correction, and (4) execution strategy

Table 1. Data Warehouse Schema Definition

Table	Column	Data Type	Description
Fact Tables			
FACT_COLLISIONS	collision_pk	SERIAL	Primary key
	collision_id	VARCHAR(20)	NYC Open Data identifier
	date_fk	INTEGER	Foreign key to DIM_DATE
	time_fk	INTEGER	Foreign key to DIM_TIME
	location_fk	INTEGER	Foreign key to DIM_LOCATION
	total_injured	INTEGER	Total persons injured
	total_killed	INTEGER	Total fatalities
	pedestrians_injured/killed	INTEGER	Pedestrian casualties
	cyclists_injured/killed	INTEGER	Cyclist casualties
	motorists_injured/killed	INTEGER	Motorist casualties
FACT_TRAFFIC_VOLUME	volume_pk	SERIAL	Primary key
	date_fk, time_fk, location_fk	INTEGER	Dimension foreign keys
	vehicle_count	INTEGER	Hourly vehicle count
	segment_id	VARCHAR(20)	Road segment identifier
Dimension Tables			
DIM_DATE	date_pk	SERIAL	Primary key
	full_date, year, quarter, month	DATE/INT	Calendar attributes
	week_of_year, day_of_week	INTEGER	Week/day identifiers
	is_weekend, is_holiday	BOOLEAN	Temporal flags
DIM_TIME	time_pk	SERIAL	Primary key
	hour, minute	INTEGER	Time components
	part_of_day	VARCHAR(20)	Morning/Afternoon/Evening/Night
	is_rush_hour	BOOLEAN	Rush hour flag (7–9, 16–19)
DIM_LOCATION	location_pk	SERIAL	Primary key
	borough, zip_code	VARCHAR	Geographic identifiers
	latitude, longitude	DECIMAL(10,7)	GPS coordinates
	roadway_name, cross_street	VARCHAR(100)	Street identifiers
DIM_VEHICLE_TYPE	vehicle_type_pk	SERIAL	Primary key
	vehicle_type, vehicle_category	VARCHAR	Type classification
DIM_CONTRIBUTING_FACTOR	factor_pk	SERIAL	Primary key
	factor_description, factor_category	VARCHAR	Factor classification

planning determines the optimal execution order and result aggregation approach.

3.4 LLM Prompt Engineering

The effectiveness of query translation depends critically on prompt design. Our system prompt incorporates four key components:

1. **Role Definition**: Establishes the LLM as a database query translator for traffic safety analytics
2. **Schema Context**: Complete schema definitions including table names, column names, data types, and foreign key relationships
3. **Output Format**: JSON structure specification for query plans with required fields
4. **Translation Guidelines**: Rules for database selection, join strategies, and aggregation patterns

Listing 1 shows an abbreviated version of the prompt template.

4 Experimental Methodology

Our experimental evaluation uses the NYC Open Data traffic datasets, comprising motor vehicle collision records and traffic volume measurements. Table 2 summarizes the dataset characteristics after ETL processing.

The ETL pipeline implements several data quality mechanisms: (1) defensive NaN/NULL handling for pandas-to-SQL type conversion, (2) transaction management with batch commits and automatic rollback, (3) dimension deduplication using `IS NOT DISTINCT FROM` for NULL-safe comparison, and (4) progress tracking with error counting and continuation semantics.

Table 2. Dataset Statistics After ETL Processing

Metric	Value
Collision Records	50,000
Traffic Volume Records	111,326
Total Fact Table Records	161,326
Unique Locations	18,411
Date Range	2020–2024
Date Dimension Entries	704
Time Dimension Entries	1,440
Vehicle Types	152
Contributing Factors	61
PostgreSQL Database Size	142 MB
MongoDB Collection Size	89 MB

The data loading process required careful handling of NYC Open Data inconsistencies. Collision records exhibited varying completeness across years, with older records lacking precise coordinates or contributing factor details. We implemented a multi-pass ETL strategy: (1) initial bulk load with permissive NULL handling, (2) dimension table population with deduplication, (3) fact table foreign key resolution with fallback to "Unknown" dimension

Listing 1. Query Translation Prompt Template (Abbreviated)

```
SYSTEM_PROMPT = """
You are a query translator for a traffic analytics system.

DATABASE SCHEMA:
- PostgreSQL (Star Schema):
  FACT_COLLISIONS(collision_pk, date_fk, time_fk, location_fk, total_injured, total_killed,
      pedestrians_injured, ...)
  DIM_DATE(date_pk, full_date, year, month, quarter, week_of_year, day_of_week, is_weekend, is_holiday)
  DIM_LOCATION(location_pk, borough, zip_code, latitude, longitude, roadway_name, cross_street)
  DIM_TIME(time_pk, hour, minute, part_of_day, is_rush_hour)

- MongoDB: Raw collision documents with flexible schema
- Redis: Cached aggregations (borough_counts, hourly_stats)

OUTPUT FORMAT (JSON):
{
  "query_type": "simple_analytical|time_series|complex_analytical|geospatial|comparative|correlation",
  "databases": ["postgresql_warehouse"],
  "strategy": "warehouse_first|cache_first",
  "postgresql_query": {"query": "SELECT ..."},
  "explanation": "Brief reasoning for database selection and query approach"
}

RULES:
1. Use PostgreSQL for all aggregations and dimensional analysis
2. Use proper JOIN syntax with dimension tables (e.g., fc.location_fk = dl.location_pk)
3. Include GROUP BY for all non-aggregated columns in SELECT
4. Use Redis cache for frequently requested aggregations
"""
```

Algorithm 1 LLM-Powered Query Translation

Require: Natural language query q, Schema context S, LLM model M
Ensure: Query execution plan P with database-specific queries

Phase 1: Query Classification
$prompt_1 \leftarrow$ BuildClassificationPrompt(q, S)
$classification \leftarrow M$.Generate($prompt_1$)
$query\_type \leftarrow$ ParseQueryType($classification$)
$target\_dbs \leftarrow$ IdentifyTargetDatabases($classification$)
Phase 2: Query Plan Generation
$prompt_2 \leftarrow$ BuildTranslationPrompt(q, S, $query\_type$)
$raw\_plan \leftarrow M$.Generate($prompt_2$)
$P \leftarrow$ ParseQueryPlan(raw_plan)
Phase 3: Validation and Correction
for each db_query in $P.queries$ **do**
  $valid \leftarrow$ ValidateSchema(db_query, S)
  if not $valid$ **then**
    $correction\_prompt \leftarrow$ BuildCorrectionPrompt(db_query, S)
    $db\_query \leftarrow M$.Generate($correction\_prompt$)
  end if
end for
Phase 4: Execution Strategy
$P.strategy \leftarrow$ DetermineExecutionOrder($P.queries$, $target\_dbs$)
$P.aggregation \leftarrow$ PlanResultAggregation($query\_type$)
return P

entries, and (4) index creation on frequently-joined columns. The complete ETL pipeline executed in approximately 12 minutes on commodity hardware, achieving a throughput of approximately 13,400 records per minute.

We designed a comprehensive query suite spanning six categories with 39 total queries. Table 3 provides representative examples from each category.

To characterize the accuracy-latency trade-off, we evaluate two Google Gemini model variants under identical conditions. We selected models from the same family to isolate the effect of model capacity on translation quality while controlling for architectural differences, training data, and API behavior:

- **Gemini 3 Pro**: Larger model optimized for complex reasoning tasks
- **Gemini 3 Flash**: Smaller, faster model designed for low-latency applications

Both models use identical configuration: temperature 0.2, top-p 0.95, max tokens 8192. Each query is evaluated independently with fresh context to prevent cross-query learning effects.

Translation accuracy is assessed using a three-tier classification scheme with explicit criteria:

Fully Correct ($score = 1.0$): The generated query satisfies all conditions:

- Correctly identifies target database(s)
- Includes all necessary table joins
- Applies appropriate aggregation functions
- Contains correct WHERE clause predicates
- Produces semantically equivalent results

Table 3. Query Suite Examples by Category

Category	N	Example Query	Expected SQL Pattern
Simple Analytical	10	"How many collisions happened in Manhattan last month?"	`SELECT COUNT(*) FROM fact_collisions fc JOIN dim_location dl ON ... WHERE dl.borough = 'MANHATTAN'`
Time Series	8	"Show the trend of pedestrian collisions by month for 2024"	`SELECT dd.month, SUM(fc.pedestrians_injured) FROM ... GROUP BY dd.month ORDER BY dd.month`
Complex Analytical	6	"Break down Manhattan collisions by vehicle type and time of day"	`SELECT dvt.vehicle_type, dt.part_of_day, COUNT(*) FROM ... GROUP BY dvt.vehicle_type, dt.part_of_day`
Geospatial	5	"Find dangerous intersections within 2 miles of Central Park"	`SELECT dl.roadway_name, COUNT(*) FROM ... WHERE latitude BETWEEN ... GROUP BY ...`
Comparative	6	"Compare collision rates between morning and evening rush hours"	`SELECT dt.part_of_day, COUNT(*) FROM ... WHERE dt.is_rush_hour GROUP BY dt.part_of_day`
Correlation	4	"Is there a relationship between traffic volume and collision frequency?"	`SELECT CORR(avg_volume, collision_count) FROM (subquery joins)`

Partially Correct ($score = 0.5$): The query captures primary intent but has minor issues such as missing optional filters, suboptimal join order, or column name variations requiring minor correction.

Incorrect ($score = 0.0$): The query fails to capture intent, targets wrong tables, or contains fundamental logical errors.

The overall translation accuracy for category c is computed as:

$$\text{Accuracy}_c = \frac{\sum_{o=1}^{N_c} score_o}{N_c} \times 100\% \quad (1)$$

where N_c is the number of queries in category c.

5 Experimental Results

This section presents comprehensive experimental results from our evaluation of the LLM-powered polyglot persistence system on the NYC Traffic BI domain, including detailed comparison between Gemini 3 Pro and Gemini 3 Flash. As defined in Section 4, we classify translations as *fully correct* when the generated query is executable and semantically equivalent to the expected output, *partially correct* when the query captures the primary intent but requires minor corrections (e.g., column name adjustments), and *incorrect* when the query fails to capture the user's intent or contains fundamental logical errors.

5.1 Translation Accuracy: Gemini 3 Pro

Table 4 summarizes translation accuracy for Gemini 3 Pro across query categories.

Table 4. Query Translation Accuracy by Category – Gemini 3 Pro

Category	Correct (%)	Partial (%)	Incorrect (%)	N
Simple Analytical	80.0	20.0	0.0	10
Time Series	87.5	12.5	0.0	8
Complex Analytical	66.7	33.3	0.0	6
Geospatial	100.0	0.0	0.0	5
Comparative	100.0	0.0	0.0	6
Correlation	50.0	50.0	0.0	4
Overall	**82.1**	**17.9**	**0.0**	**39**

Gemini 3 Pro achieves 82.1% fully correct translations with 100% combined accuracy (fully correct plus partially correct, meaning zero translations were classified as incorrect). Notably, geospatial and comparative queries achieve perfect accuracy, while correlation queries present the greatest challenge with only 50% fully correct translations. For instance, when asked "Is there a correlation between traffic volume and collision severity?", the model correctly identified the need for statistical analysis but generated a query using `AVG()` instead of PostgreSQL's `CORR()` function, capturing the analytical intent while missing the precise statistical operation. Figure 2 visualizes this distribution across all six query categories.

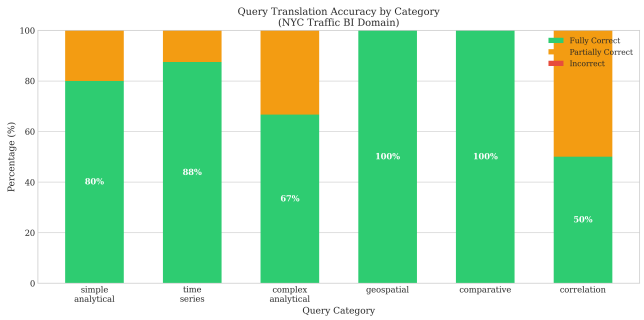

Figure 2. Query translation accuracy by category for Gemini 3 Pro showing the distribution of fully correct, partially correct, and incorrect translations.

5.2 Translation Accuracy: Gemini 3 Flash

Table 5 summarizes translation accuracy for Gemini 3 Flash.

Table 5. Query Translation Accuracy by Category – Gemini 3 Flash

Category	Correct (%)	Partial (%)	Incorrect (%)	N
Simple Analytical	80.0	20.0	0.0	10
Time Series	87.5	12.5	0.0	8
Complex Analytical	66.7	33.3	0.0	6
Geospatial	100.0	0.0	0.0	5
Comparative	100.0	0.0	0.0	6
Correlation	0.0	100.0	0.0	4
Overall	**76.9**	**23.1**	**0.0**	**39**

Gemini 3 Flash achieves 76.9% fully correct translations, approximately 5 percentage points lower than Gemini 3 Pro. The primary difference emerges in correlation queries, where Flash produces only partial translations (0% fully correct vs. 50% for Pro), as illustrated in Figure 3.

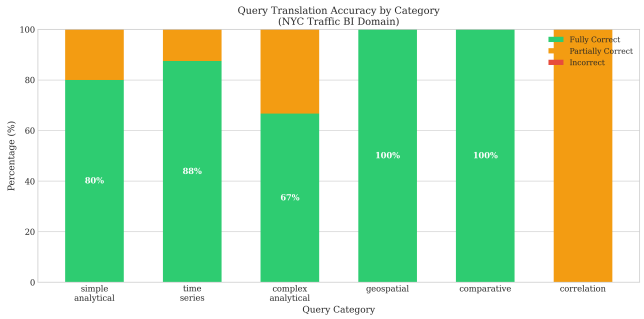

Figure 3. Query translation accuracy by category for Gemini 3 Flash.

5.3 Model Comparison: Accuracy vs. Latency Trade-off

Table 6 provides a direct comparison between the two model variants.

The comparison reveals a clear accuracy-latency trade-off:

- **Accuracy**: Pro achieves 5.2 percentage points higher fully correct accuracy
- **Latency**: Flash provides 1.76× speedup (10.1s vs. 17.8s average)
- **Robustness**: Both achieve 100% combined accuracy

Table 6. Model Comparison: Gemini 3 Pro vs. Flash

Metric	Gemini 3 Pro	Gemini 3 Flash
Fully Correct (%)	82.1	76.9
Partially Correct (%)	17.9	23.1
Incorrect (%)	0.0	0.0
Combined Accuracy (%)	100.0	100.0
Avg. Translation Time (s)	17.8	10.1
Min Translation Time (s)	12.9	9.0
Max Translation Time (s)	26.5	11.5
Speedup Factor	1.0×	1.76×

5.4 Error Category Analysis

Analysis of partial translations reveals systematic patterns. Table 7 categorizes primary error types.

Table 7. Error Categories in Query Translation

Error Category	Pro (%)	Flash (%)
Schema Misalignment	45.2	52.1
Missing Aggregation	18.3	21.4
Incorrect Join Logic	12.1	14.8
Temporal Filter Errors	15.6	8.2
Statistical Function Gaps	8.8	3.5

Schema Misalignment: The most common error involves generating column names that differ from the actual schema (e.g., `date_sk` instead of `date_pk`). This error is purely syntactic and does not affect semantic correctness.

Statistical Function Gaps: Flash rarely attempts advanced statistical functions (CORR, STDDEV), instead falling back to simpler aggregations, explaining its lower correlation query accuracy.

To illustrate these error patterns, consider the correlation query "Is there a relationship between traffic volume and collision frequency?" Gemini 3 Pro correctly generated a query using PostgreSQL's `CORR()` function with properly structured CTEs for aggregating both traffic volumes and collision counts by location before computing the correlation coefficient. In contrast, Gemini 3 Flash produced a query that grouped data by borough and returned raw counts without statistical correlation, capturing the analytical intent but missing the statistical depth, a characteristic example of the "Statistical Function Gaps" error category.

5.5 Performance Analysis

Tables 8 and 9 present processing times by category.

Table 8. Query Processing Performance – Gemini 3 Pro (ms)

Category	Translation	Execution	Total
Simple Analytical	20,434	8	20,442
Time Series	12,884	0	12,884
Complex Analytical	15,469	0	15,469
Geospatial	26,506	0	26,506
Comparative	15,452	0	15,452
Correlation	15,887	27	15,914

Database execution time remains negligible (<55ms) for both models, confirming that LLM translation dominates

Table 9. Query Processing Performance – Gemini 3 Flash (ms)

Category	Translation	Execution	Total
Simple Analytical	10,118	9	10,127
Time Series	9,046	0	9,046
Complex Analytical	9,751	0	9,751
Geospatial	11,474	0	11,474
Comparative	9,478	0	9,478
Correlation	10,645	55	10,700

Table 10. Database Execution Performance (Valid Queries)

Category	Avg (ms)	Max (ms)	Rows
Simple Analytical	8.6	12.3	1–5
Time Series	4.2	8.1	12–24
Complex Analytical	15.3	28.7	10–50
Geospatial	22.1	45.2	5–20
Comparative	6.8	11.4	2–5
Correlation	54.9	67.3	1

overall latency (>99%). Execution times of 0ms indicate sub-millisecond query execution, rounded down from values below the measurement threshold (1ms precision).

While Tables 8 and 9 provide precise timing measurements, Figures 4 and 5 visualize the dramatic disproportion between translation and execution times, execution bars are barely visible, immediately conveying that LLM latency dominates the system.

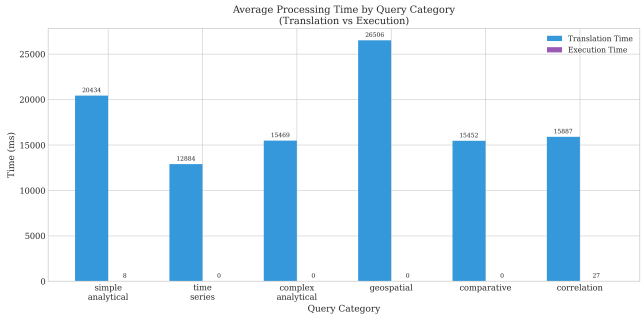

Figure 4. Average processing time by category for Gemini 3 Pro showing translation time (dominant) versus execution time (negligible).

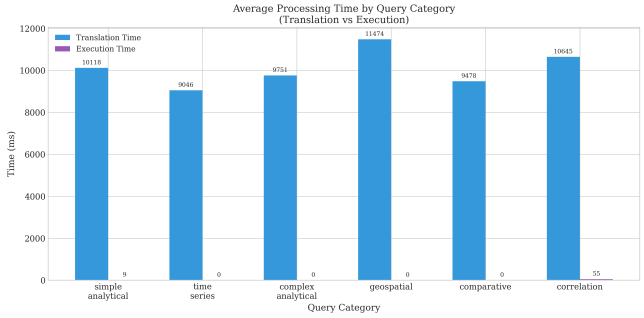

Figure 5. Average processing time by category for Gemini 3 Flash demonstrating reduced translation latency.

5.6 Database Execution Performance

Table 10 presents detailed database performance for valid queries.

Correlation queries exhibit the highest execution time (54.9ms) due to statistical aggregations, but even the most expensive queries complete in under 70ms.

The stark contrast between translation time (seconds) and execution time (milliseconds) suggests that the polyglot persistence architecture itself introduces negligible overhead. Even complex correlation queries involving joins across fact tables complete in under 70ms, demonstrating that PostgreSQL's query optimizer effectively handles the star schema workload. This finding has important implications for system design: optimization efforts should focus on the LLM translation layer rather than database tuning, and query result caching at the translation layer can provide substantial latency improvements for repeated query patterns.

Table 6 summarizes overall system performance across key dimensions. Both models achieve high SQL validity and execution success (near 100%), with the primary differentiation appearing in translation accuracy and response time, Pro excels in accuracy while Flash demonstrates superior response time performance.

6 Cross-Domain Validation

To validate the generalizability of our framework, we compare Traffic/BI results with prior work that applied the same polyglot persistence architecture to a social network application (de Curtò and de Zarzà 2025), however in this study we use the variant Gemini 3 Pro for the comparison. That study implemented a graph-centric system for managing user profiles, friendships, and community interactions, primarily leveraging Neo4j for relationship traversals with MongoDB and Redis as supporting backends. By contrasting the graph-oriented Social Network domain with our aggregation-intensive Traffic/BI domain, we can assess how effectively the LLM-powered architecture adapts across fundamentally different data models and query patterns.

6.1 Domain Characteristics

The two domains exhibit distinct characteristics:

Social Network Domain: Relationship-centric with deep graph traversals, primarily leveraging Neo4j for friend-of-friend queries and community detection. Approximately 600 records across user profiles and connections.

Traffic/BI Domain: Aggregation-intensive with dimensional modeling, centering on PostgreSQL's star schema for collision analysis. Approximately 161,000 records across collision events and traffic volumes. Figure 6 contrasts these domain characteristics across six dimensions, highlighting how social networks emphasize relationship depth while Traffic/BI prioritizes data volume and aggregation intensity.

6.2 Query Complexity Metrics

Table 11 quantifies translation complexity differences.

Traffic/BI queries average 2.8× longer and reference 1.8× more tables, explaining the 4–8× latency increase.

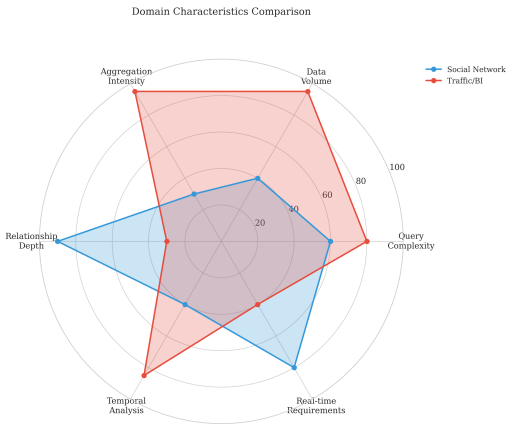

Figure 6. Radar chart comparing domain characteristics. Social networks emphasize relationship depth; Traffic/BI prioritizes data volume and aggregation.

Table 11. Query Complexity Metrics by Domain

Metric	Social	Traffic/BI
Avg. Query Length (tokens)	45	127
Avg. JOIN Operations	0.8	2.4
Avg. Aggregation Functions	0.3	1.8
Schema Tables Referenced	2.1	3.8
Prompt Context Size (tokens)	1,200	3,400

6.3 Architecture Adaptation

The polyglot persistence framework dynamically adapts its database utilization patterns based on domain requirements. Figure 7 illustrates the contrasting database usage distributions: the Social Network domain routes 60% of queries to Neo4j for graph traversals, while Traffic/BI directs 75% of queries to PostgreSQL for analytical aggregations. This adaptation demonstrates the framework's flexibility in leveraging specialized backends according to workload characteristics.

6.4 Translation Accuracy Comparison

Figure 8 presents the accuracy comparison between domains, revealing Traffic/BI's substantial improvement over the Social Network baseline.

Traffic/BI achieves substantially higher accuracy due to:

- **Query Structure**: SQL queries align with LLM training data
- **Schema Clarity**: Star schema provides clearer context
- **Reduced Ambiguity**: Dimensional queries have more deterministic translations

These findings align with broader observations in the text-to-SQL literature, where LLMs demonstrate stronger performance on well-structured relational schemas compared to graph query languages. The star schema's explicit foreign key relationships and standardized naming conventions (e.g., `_pk` for primary keys, `_fk` for foreign keys) provide unambiguous context that reduces translation errors. Conversely, Cypher queries for Neo4j require reasoning about variable-length path patterns and relationship types that are less represented in typical LLM training corpora. Figure 9 provides a detailed breakdown by query category, showing that simpler query types achieve comparable accuracy across domains while complex analytical queries benefit most from the structured dimensional schema.

6.5 Cross-Domain Summary

Table 12 summarizes the comprehensive comparison.

Table 12. Cross-Domain Comparison of Polyglot Persistence Framework

Characteristic	Social Network	Traffic/BI
Data Architecture		
Primary Database	Neo4j (Graph)	PostgreSQL (Star Schema)
Secondary Databases	MongoDB, Redis	MongoDB, Neo4j, Redis
Data Model	Property Graph	Dimensional Model
Total Records	∼600	∼161,000
Query Characteristics		
Primary Query Type	Relationship Traversal	Aggregation/Analytics
Query Categories	3	6
Queries Tested	15	39
LLM Translation Performance (Gemini 3 Pro)		
Fully Correct (%)	60.0	82.1
Partially Correct (%)	28.3	17.9
Combined Accuracy (%)	88.3	100.0
Avg. Translation Time (ms)	3,333	17,772

Figure 10 illustrates the substantial difference in evaluation scale between domains using a logarithmic axis: Traffic/BI represents a 269× increase in data volume and 2.6× expansion in query coverage compared to the Social Network baseline. Figure 11 demonstrates the corresponding impact on translation time, with Traffic/BI requiring 4–8× longer translation across all complexity levels due to the larger schema context and more sophisticated query patterns.

7 Discussion

7.1 Key Findings

Our experimental evaluation reveals several important findings:

Accuracy-Latency Trade-off: Gemini 3 Pro achieves 5.2 percentage points higher accuracy but requires 1.76× longer processing. This trade-off has practical implications: latency-sensitive applications may prefer Flash, while accuracy-critical applications benefit from Pro.

Cross-Domain Generalizability: Translation accuracy improves from 60.0% to 82.1% with dimensional schemas, suggesting LLMs perform better with structured, well-normalized data models aligned with SQL training data.

Strong Semantic Understanding: Both models achieve 100% combined accuracy with zero incorrect translations, demonstrating robust intent capture even when failing to generate perfectly executable queries.

Category-Specific Performance: Both models excel at geospatial and comparative queries but struggle with correlation analysis. Flash shows more pronounced difficulty with statistical reasoning (0% vs. 50% fully correct on correlation).

Latency Dominated by Translation: LLM calls account for >99% of total latency, presenting clear optimization opportunities through caching, batching, or model distillation.

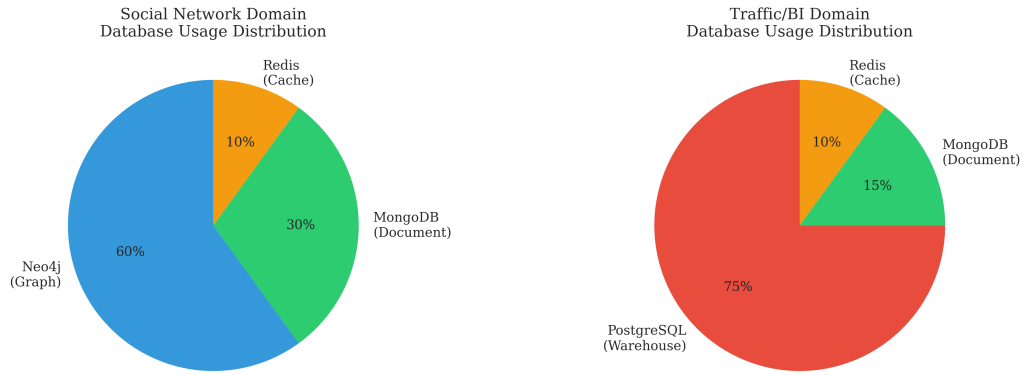

Figure 7. Database usage distribution. Social Network: 60% Neo4j; Traffic/BI: 75% PostgreSQL.

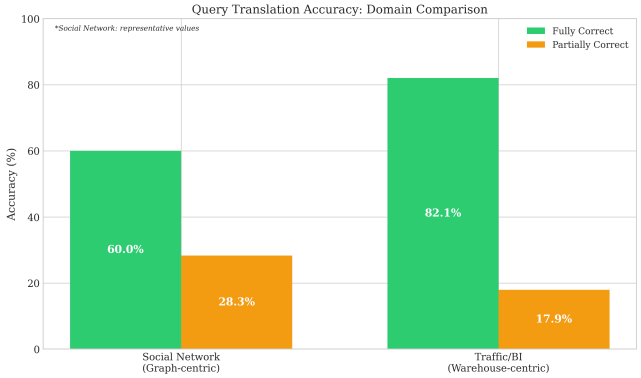

Figure 8. Query translation accuracy comparison. Traffic/BI achieves higher accuracy (82.1% vs. 60.0%).

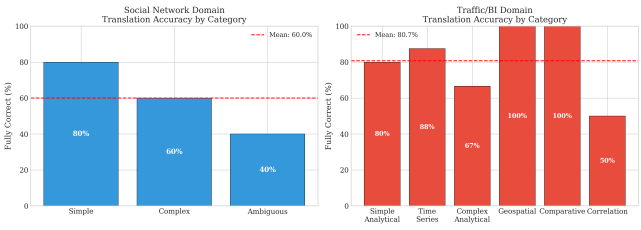

Figure 9. Translation accuracy by category across domains.

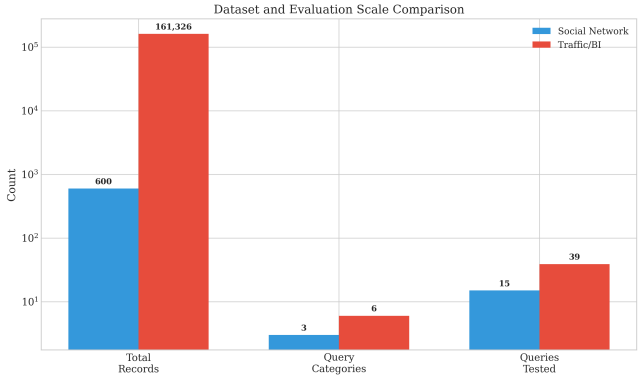

Figure 10. Dataset and evaluation scale comparison (log scale). Traffic/BI: 269× data increase, 2.6× query coverage.

7.2 Cost-Benefit Analysis

Table 13 estimates API costs based on December 2024 Gemini pricing.

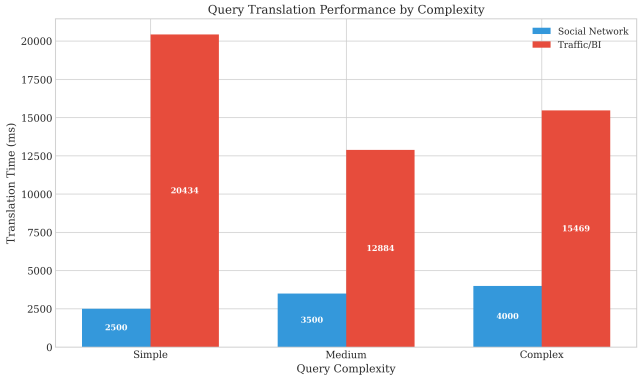

Figure 11. Translation time by complexity. Traffic/BI requires 4–8× longer translation.

Table 13. Estimated API Cost per Query

Component	Pro	Flash
Input Tokens (avg)	3,400	3,400
Output Tokens (avg)	450	380
Cost per Query	$0.0068	$0.0004
Cost per 1000 Queries	$6.80	$0.40

Flash provides approximately 17× cost reduction, representing significant operational savings for high-volume applications.

7.3 Implementation Considerations

Based on implementation experience, we identify practical considerations:

Rate Limiting: Our implementation incorporates adaptive rate limiting with exponential backoff (2–60 seconds based on consecutive errors). Production deployments should implement request queuing and caching.

Schema Context Optimization: Selective schema inclusion can reduce prompt size by 40–60% without sacrificing accuracy for simple queries, though complex queries benefit from full context.

Error Recovery: The pipeline implements graceful degradation with JSON parsing fallbacks and iterative correction prompts for validation failures.

Caching Strategy: Implementing semantic cache for query patterns can dramatically improve response times. We recommend caching at the query plan level to enable partial cache hits.

7.4 Practical Recommendations

- For **interactive applications** requiring sub-15-second responses, Flash provides acceptable accuracy with reduced latency
- For **batch processing** or accuracy-critical applications, Pro offers superior translation quality
- For **correlation/statistical queries**, larger models should be preferred
- **Schema context enhancement** should be prioritized to improve SQL validity

7.5 Implications for Semantic Web Systems

Our findings have direct implications for Semantic Web query interfaces. The observation that LLMs achieve higher accuracy on well-structured relational schemas (82.1%) compared to graph queries (60.0%) suggests that SPARQL generation may face similar challenges to Cypher, given both involve graph pattern matching. However, RDF's standardized vocabulary and explicit typing could provide richer context than Neo4j's property graphs, potentially improving translation accuracy. The semantic similarity between our dimensional model's foreign key relationships and RDF's predicate-based linking suggests that hybrid architectures, combining relational warehouses with knowledge graph overlays, could leverage LLMs effectively for both analytical aggregations and semantic reasoning. Our prompt engineering approach, which embeds complete schema context, parallels ontology-aware prompting strategies emerging in the SPARQL generation literature.

7.6 Threats to Validity

Internal Validity: Query classification involves subjective judgment (mitigated through explicit criteria); LLM responses exhibit non-determinism (mitigated with temperature 0.2).

External Validity: Results are specific to NYC traffic domain; query suite may not cover all production patterns; LLM capabilities evolve rapidly.

Construct Validity: Translation accuracy measures semantic correctness but not user satisfaction; execution time excludes network latency to LLM APIs.

7.7 Reproducibility

To ensure reproducibility:

- **LLM Configuration**: Temperature 0.2, top-p 0.95, max tokens 8192
- **Database Versions**: PostgreSQL 14, MongoDB 8.0, Neo4j 5.x, Redis 6.0
- **Environment**: Python 3.10, Ubuntu 22.04, 16GB RAM

To facilitate independent replication, we provide the complete experimental infrastructure including: (1) ETL scripts for NYC Open Data ingestion, (2) schema creation scripts for all four database backends, (3) the complete 39-query evaluation suite with expected outputs, (4) evaluation harness with automated scoring, and (5) visualization generation scripts. The evaluation can be reproduced with one command after configuring database connections and API credentials.

8 Conclusion

This paper has presented a comprehensive evaluation of integrating polyglot persistence architectures with Large Language Models for business intelligence applications. Our system, combining PostgreSQL data warehouses, MongoDB document stores, Neo4j graph databases, and Redis caches, was evaluated using two Google Gemini model variants to characterize the accuracy-latency trade-off.

The key contributions include:

1. A generalizable architecture extending polyglot persistence with LLMs from social networks to business intelligence domains
2. A comprehensive evaluation framework with six query categories and 39 queries
3. Empirical characterization of the accuracy-latency trade-off: Pro achieves 82.1% accuracy with 17.8s latency; Flash achieves 76.9% with 10.1s (1.76× speedup)
4. Cross-domain validation demonstrating improved accuracy (60.0% to 82.1%) with dimensional schemas and 269× data scale increase
5. Evidence that both models achieve 100% combined accuracy with zero incorrect translations
6. Identification of challenges (schema alignment, correlation queries) and optimization opportunities (caching, prompt engineering, hybrid routing)

As LLM capabilities advance, we anticipate the accuracy-latency trade-off will become less pronounced, enabling real-time natural language database interfaces without sacrificing quality. The architecture and methodology presented here provide a foundation for continued research toward accessible sophisticated data analysis.

8.1 Future Directions

Several promising directions emerge from this work. First, fine-tuning smaller models on domain-specific query translation tasks could achieve Flash-level latency with Pro-level accuracy, reducing both cost and latency. Second, hybrid routing strategies could dynamically select between Pro and Flash based on detected query complexity, using Flash for simple aggregations and Pro for statistical analyses. Third, schema-aware prompt compression techniques could reduce context size while preserving translation accuracy, addressing the 3,400-token prompt overhead observed in our experiments. Finally, query plan caching with semantic similarity matching could enable partial cache hits for semantically similar but lexically different queries, dramatically reducing average response times for production workloads.

Acknowledgements

The authors thank the BARCELONA Supercomputing Center for access to MareNostrum 5 and technical support. This research was supported by the LUXEMBOURG Institute of Science and Technology (LIST) through the projects "ADIALab-MAST" and "LLMs4EU". The LLMs4EU project is co-funded by the European Union under the Digital Europe Programme, Grant Agreement No 101198470. Additional support was provided by the BARCELONA Supercomputing Center through the project "TIFON", funded by the Centre for the Development of Industrial Technology (CDTI) with the support of the Spanish Ministry of Science and Innovation under file number MIG-20232039, and by the Agencia Estatal de Investigación (AEI).

Declaration of conflicting interests

The authors declare no conflict of interest.

Supplemental material

Implementation available at: https://github.com/drdecurto/polyglot-persistence-llm

References

Amer-Yahia S, Bonifati A, Chen L, Li G, Shim K, Xu J and Yang X (2023) From large language models to databases and back: A discussion on research and education. *ACM SIGMOD Record* 52(3): 49–56.

Brown T, Mann B, Ryder N et al. (2020) Language models are few-shot learners. In: *Advances in Neural Information Processing Systems*, volume 33. pp. 1877–1901.

Carlson J (2013) *Redis in action*. Simon and Schuster.

Chauhan A (2019) A review on various aspects of mongodb databases. *International Journal of Engineering Research & Technology (IJERT)* 8(05): 90–92.

Chen H and 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.

Da Silva MD and Tavares HL (2015) *Redis Essentials*. Packt Publishing Ltd.

de Curtò J and 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.

de Curtò J, de Zarzà I and Calafate CT (2025) Integrating polyglot persistence with large language models for scalable social network applications. *Procedia Computer Science* 270: 733–743. DOI:https://doi.org/10.1016/j.procs.2025.09.193. URL https://www.sciencedirect.com/science/article/pii/S1877050925028637. 29th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2025).

Deka GC (2018) Nosql polyglot persistence. In: *Advances in Computers*, volume 109. Elsevier, pp. 357–390.

Grolinger K, Higashino WA, Tiwari A and Capretz MA (2013) Data management in cloud environments: Nosql and newsql data stores. *Journal of Cloud Computing: advances, systems and applications* 2: 1–24.

Győrödi C, Győrödi R, Pecherle G and 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.

Hogan A, Blomqvist E, Cochez M, d'Amato C, Melo GD, Gutierrez C, Kirrane S, Gayo JEL, Navigli R, Neumaier S et al. (2021) Knowledge graphs. *ACM Computing Surveys (Csur)* 54(4): 1–37.

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.

Li Y and 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.

Miller JJ (2013) Graph database applications and concepts with neo4j. In: *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*, volume 2324. pp. 141–147.

Nascimento ER, García G, Izquierdo YT, Feijó L, Coelho GM, de Oliveira AR, Lemos M, Garcia RL, Leme LAP and Casanova MA (2025) Llm-based text-to-sql for real-world databases. *SN Computer Science* 6(2): 130.

Oliveira FR and 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.

Team G, Anil R, Borgeaud S, Alayrac JB, Yu J, Soricut R, Schalkwyk J, Dai AM, Hauth A, Millican K et al. (2023) Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* .

Van Landuyt D, Benaouda J, Reniers V, Rafique A and 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.

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł and Polosukhin I (2017) Attention is all you need. In: *Advances in Neural Information Processing Systems*, volume 30. pp. 5998–6008.

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.

Yang S, Teng M, Dong X and Bo F (2023) Llm-based sparql generation with selected schema from large scale knowledge base. In: *China conference on knowledge graph and semantic computing*. Springer, pp. 304–316.

Zhou X, Sun Z and Li G (2024) Db-gpt: Large language model meets database. *Data Science and Engineering* 9(1): 102–111.