International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 02 | Feb 2026 www.irjet.net p-ISSN: 2395-0072

AI-Powered Conversational Assistant for Complex & Secure database Interaction and Intelligent Backend Monitoring.

Shravani .B. Dumbre[1], Bandla Laxmi Prasanna[2], Swati .S. Dhande[3], prof .Sangita Bhoyar[4]

[1,2,3,] *Student Computer Engineering Department,V. E. S. Polytechnic, Mumbai, India*
[4] *Professor, , Computer Engineering Department, V. E. S. Polytechnic Mumbai, India*

Abstract - *This paper introduces Querion, an AI-powered conversational assistant designed for complex and secure database interaction coupled with intelligent backend monitoring for application services. Organizations and users face challenges in accessing enterprise data and observing backend behavior, with existing solutions often requiring technical expertise or lacking integrated security and real-time observability. The proposed system addresses these by leveraging Large Language Models (LLMs) via the Groq API for fast inference and LangChain for secure NL→SQL chaining, translating natural language queries into accurate SQL commands while enforcing Role-Based Access Control (RBAC), query sandboxing, end-to-end encryption, NVIDIA NeMo Guardrails for prompt injection prevention, and LLM Guard for real-time input/output scanning against OWASP LLM Top 10 threats. Additionally, the intelligent backend monitoring module enables real-time connection to user backend applications or projects through IDE extensions (e.g., VS Code) or WebSockets, capturing terminal output, console errors, API endpoint usage, runtime exceptions, and application logs. These events are visualized via interactive dashboards and charts, with AI-powered analysis using Groq API and LangChain providing natural language explanations of errors, highlighting efficient patterns (e.g., optimal API responses), and suggesting corrective code prompts to support rapid debugging, troubleshooting, performance optimization, and learning across diverse user scenarios. This unified approach delivers secure, transparent, and efficient data access alongside comprehensive backend observability, empowering non-technical users, developers, students, and organizations toward advanced AI-driven database management and application maintenance.*

Key Words: AI-powered conversational assistant, natural language to SQL (NL→SQL), large language models (LLMs), enterprise database security, role-based access control (RBAC), prompt injection prevention, intelligent backend monitoring, real-time observability, runtime error analysis, IDE integration, WebSocket streaming, visual analytics dashboards, developer debugging, Groq API, LangChain, NVIDIA NeMo Guardrails, LLM Guard, FastAPI, Chainlit, VS Code extension.

1. INTRODUCTION

Modern organizations rely on diverse and large-scale data sources, yet traditional database interaction methods such as manual SQL querying remain inaccessible to non-technical users and prone to inefficiencies and security risks. The growing demand for data democratization and self-service business intelligence (BI) necessitates secure, scalable, and user-friendly solutions [24].

Conversational AI systems powered by NLP and Large Language Models (LLMs) enable natural language to SQL translation, improving accessibility and accuracy through advanced text-to-SQL techniques [1], [3], [4], [7], [26]–[28]. However, LLM-integrated systems face security threats such as prompt injection and guardrail bypass attacks [9]–[13], requiring robust defenses including RBAC, encryption, parameterized queries, and NeMo Guardrails aligned with OWASP protections [5], [25]. Additionally, real-time backend observability is essential for application reliability [14]–[17], with WebSocket-based monitoring [18], [19] and AI-driven log analysis enhancing anomaly detection and debugging [2], [6], [29].

This research proposes Querion, an AI-powered conversational assistant that integrates secure text-to-SQL translation with intelligent backend monitoring. By combining automated query generation [1], [7], comprehensive LLM security [5], [13], and AI-enhanced runtime analysis [6], [29], Querion enables secure, efficient, and accessible data-driven decision-making and application maintenance.

2. LITERATURE REVIEW & BACKGROUND

2.1. Conversational AI in Databases

Conversational AI for databases has evolved from rule-based SQL templates [1] to ML-based translation models [2], improving accuracy but often struggling with multi-turn dialogues and cross-domain queries. Recent advances in LLMs, using platforms like Groq API and LangChain, enable low-latency NL→SQL translation with models such as LLaMA 3, Mixtral, and GPT-4o [3][4]. Tools like SQLDatabaseChain allow query generation, execution, and summarization in a single pipeline, supporting context-aware interactions. Benchmark datasets like Spider, WikiSQL, and CoSQL assess performance across complex,

cross-domain scenarios [5][6][7]. However, integrated security and real-time monitoring are still limited [8][9].

2.2. Secure Database Interaction Techniques

Secure database access requires RBAC, query sandboxing, encryption, and differential privacy to protect sensitive data [10][12]. LLM-specific threats, including prompt injections and jailbreaks [13][14], necessitate advanced defenses such as NVIDIA NeMo Guardrails [15], LLM Guard [16], and parameterized queries [17]. Most systems, however, do not fully integrate these measures, emphasizing the need for multi-layered security combining RBAC, sandboxing, guardrails, and runtime scanning [18][20].

2.3. Backend Monitoring Approaches

Backend monitoring ensures system reliability. Modern approaches use structured logging, WebSocket streaming, and IDE-integrated tools to capture terminal output, console errors, API metrics, and runtime events [21][23]. AI-powered analysis adds natural language log queries, automated error explanations, and actionable code suggestions [24][25]. Tools like OpenTelemetry, Prometheus, and Grafana enable interactive dashboards, while IDE extensions (e.g., VS Code) provide real-time, contextual insights [26][28].

2.4. Gaps in Existing Research

1. **Incomplete security integration:** Isolated safeguards without unified enforcement.
2. **Limited real-time observability:** Lack of AI monitoring for user-specific backend projects.
3. **Minimal IDE integration:** Few platforms support live log streaming and automated debugging [29][30].
4. **Model limitations:** LLMs can hallucinate queries, introduce latency, or struggle with large schemas [31][32].

These gaps highlight the need for a unified framework combining accurate NL→SQL translation, robust security, and intelligent backend monitoring

3. COMPARATIVE STUDY OF MARKET

4. PROPOSED SOLUTION: AI-POWERED CONVERSATIONAL ASSISTANT FOR COMPLEX & SECURE DATABASE INTERACTION AND INTELLIGENT BACKEND MONITORING

4.1 Core Philosophy and Approach

Querion addresses the challenges of complex database access and backend observability by combining AI-driven natural language interaction with robust security and real-time monitoring. Leveraging NLP and LLMs (Groq API, Llama 3.3, LangChain), it converts multilingual text inputs into precise SQL queries, supports multi-turn dialogues, resolves ambiguities, and enables non-technical users to perform sophisticated queries and analytics.

Security is foundational: RBAC, query sandboxing, end-to-end encryption, input validation, parameterized queries, NVIDIA NeMo Guardrails, and LLM Guard protect against SQL injection, prompt injection, jailbreaks, and OWASP LLM threats. Comprehensive auditing ensures traceability.

System	Focus	Querion Advantage
AskYourDatabase	Natural Language to SQL Querying	Adds Backend Observability and Automated Error Detection
MindsDB	Conversational Database Querying	Adds Backend Monitoring and Debugging Support
Datadog	Backend Observability and Log Monitoring	Adds Conversational Database Querying and Error Analysis
Grafana	Log Monitoring and Visualization	Adds Automated Debugging and Conversational Querying

Fig -1: Comparison of Querion with Existing Conversational Database and Observability Systems

5. PROPOSED SOLUTION: AI-POWERED CONVERSATIONAL ASSISTANT FOR COMPLEX & SECURE DATABASE INTERACTION AND INTELLIGENT BACKEND MONITORING

4.1 Core Philosophy and Approach

Querion addresses the challenges of complex database access and backend observability by combining AI-driven natural language interaction with robust security and real time monitoring. Leveraging NLP and LLMs (Groq API,

Llama 3.3, LangChain), it converts multilingual text inputs into precise SQL queries, supports multi-turn dialogues, resolves ambiguities, and enables non-technical users to perform sophisticated queries and analytics.

Security is foundational: RBAC, query sandboxing, end-to end encryption, input validation, parameterized queries, NVIDIA NeMo Guardrails, and LLM Guard protect against SQL injection, prompt injection, jailbreaks, and OWASP LLM threats. Comprehensive auditing ensures traceability and compliance for sensitive data handling.

For backend monitoring, users connect their projects via IDE extensions (e.g., VS Code, Cursor, Antigravity) or WebSockets, enabling the system to capture terminal output, console logs, runtime exceptions, API activity, and application traces. AI-driven analysis interprets errors in natural language, highlights optimal performance patterns, and suggests corrective actions. All events are visualized through interactive dashboards and charts, supporting rapid debugging, performance optimization, and learning across individual and organizational projects.

4.2 Planned Features and User Experience

Conversational AI Interface: Chainlit-based chat supports multi-turn, multilingual queries and potential voice input. Users can ask natural language questions like “Why did this runtime error occur?” or “Suggest a fix for endpoint failure.”

Database Compatibility: Optimized SQL execution via SQLAlchemy, supporting SQLite, MySQL, PostgreSQL, and Oracle.

Security & Compliance: RBAC, sandboxing, encryption, auditing, and differential privacy protect sensitive data throughout.

Intelligent Backend Monitoring: Real-time visibility into backend processes, logs, API flows, runtime errors, and performance metrics, all displayed in interactive dashboards.

Integration & Accessibility: Web-based PWA interface, Slack/Microsoft Teams integration via REST APIs, modular architecture using FastAPI, WebSockets, structlog, Docker, and Kubernetes for scalable deployment.

Developer Productivity: Reduces debugging time, enhances transparency, explains backend behavior in natural language, and combines database interaction with full-stack monitoring.

Querion provides a unified, secure, and scalable platform that empowers developers and users to query databases naturally while monitoring backend systems in real time, delivering actionable insights and improving overall productivity.

4.3. Core Features and Modules

Querion integrates a comprehensive set of intelligent modules to provide secure, real-time database interaction, developer-friendly backend monitoring, and enhanced usability for diverse users. The Natural Language Interaction Module combines a Chainlit-based conversational UI with prompt-based access, supporting multi-turn dialogues, context preservation, ambiguity clarification, and natural language reasoning. It leverages LangChain and Groq API to translate simple, multilingual prompts into optimized SQL queries, enabling seamless data retrieval without requiring SQL expertise.
The Clarification and Failed Query Handling Module automatically detects ambiguous or misunderstood prompts and requests clear confirmation from the user (e.g., "Did you mean sales by region or product?"). For failed queries, it generates a concise summary of the intended question, the actual database result (or lack thereof), and a clear explanation of the failure reason—such as insufficient access rights, missing data in the database, or invalid query structure—helping users quickly understand and correct their requests.

The Historical Query Replay Module allows users to re-execute any past prompt from conversation history with a single selection. The system automatically re-runs the query against the current (real-time updated) database state, ensuring results reflect the latest data without retyping the prompt.

The Scheduled Query and Reporting Module enables users to schedule recurring or time-specific data retrieval tasks (e.g., daily sales summary at 9 AM or weekly patient risk report). The system executes scheduled queries automatically and delivers results via the interface, email, or integrated tools, with options for continuous monitoring of particular metrics.

The Tabular Output and Data Dashboard Module formats database query results in clear tabular views and interactive visualizations (charts, graphs, dashboards) using embedded tools like Dash, providing intuitive interpretation of both query outcomes and backend monitoring insights.

The AI-Based Prompt Enhancement Suggestions Module analyzes user inputs and offers improved or more precise prompt alternatives in real time (e.g., "Try: 'Show top 10 patients with highest glucose levels by age group'"), helping users achieve better accuracy and discover effective query patterns.

The Smart Data Summaries Module simplifies complex database results into concise, meaningful insights,

highlighting key trends, aggregates, and anomalies in plain language.

The Secure Access Control Module enforces high-level data security through Role-Based Access Control (RBAC) via FastAPI-Users and JWT authentication, query sandboxing, parameterized execution with SQLAlchemy, end-to-end encryption, NVIDIA NeMo Guardrails for prompt injection prevention, and LLM Guard for real-time scanning against OWASP LLM Top 10 threats, ensuring robust protection of sensitive data.

The Official Formatted Report Generation Module automatically produces professional, structured reports (PDF/HTML) from query results or monitoring insights, suitable for documentation, compliance, and decision-making.

The Backend Project Connection Module enables users to connect their application projects via IDE extensions (e.g., VS Code) or WebSocket APIs, streaming real-time terminal output, console logs, runtime exceptions, and API endpoint activity. The Runtime Error Analysis Module uses Groq API and LangChain to detect issues, convert complex error logs into simplified, easy-to-understand explanations, and provide backend reports, UI-level insights, and step-by-step guidance for resolution. The Visual Backend Observability Module aggregates streamed events into interactive dashboards showing error trends, API flows, performance timelines, and resource utilization.

The AI-Driven Backend Guidance Module allows developers to ask natural language questions about backend issues (e.g., "What should I do about this 500 error?"), receiving detailed reports, actionable insights, and resolution recommendations directly in the conversational interface.
Collectively, these modules create a unified, secure, and intelligent platform that empowers non-technical users with intuitive database access, supports developers and students with real-time debugging and guidance, and provides organizations with transparent, compliant, and efficient data and application management.

5. CONCEPTUAL SYSTEM ARCHITECTURE & DESIGN

5.1 High-Level System Overview

Querion is designed as a modular and scalable platform that integrates AI-powered natural language processing, secure database interaction, and intelligent backend monitoring. The architecture consists of three primary layers:

User Interface Layer – A conversational chat interface that captures multilingual inputs and displays query results, backend logs, dashboards, and visual analytics.

AI & Processing Layer – Powered by Groq API and LangChain, this layer converts natural language into secure SQL queries, performs backend error analysis, and maintains multi-turn conversational context.

Database & Monitoring Layer – Executes sandboxed queries on connected databases and captures real-time backend data (logs, terminal output, API activity) via secure APIs and WebSockets.
Each component operates independently but communicates securely, ensuring modularity, maintainability, and scalability.

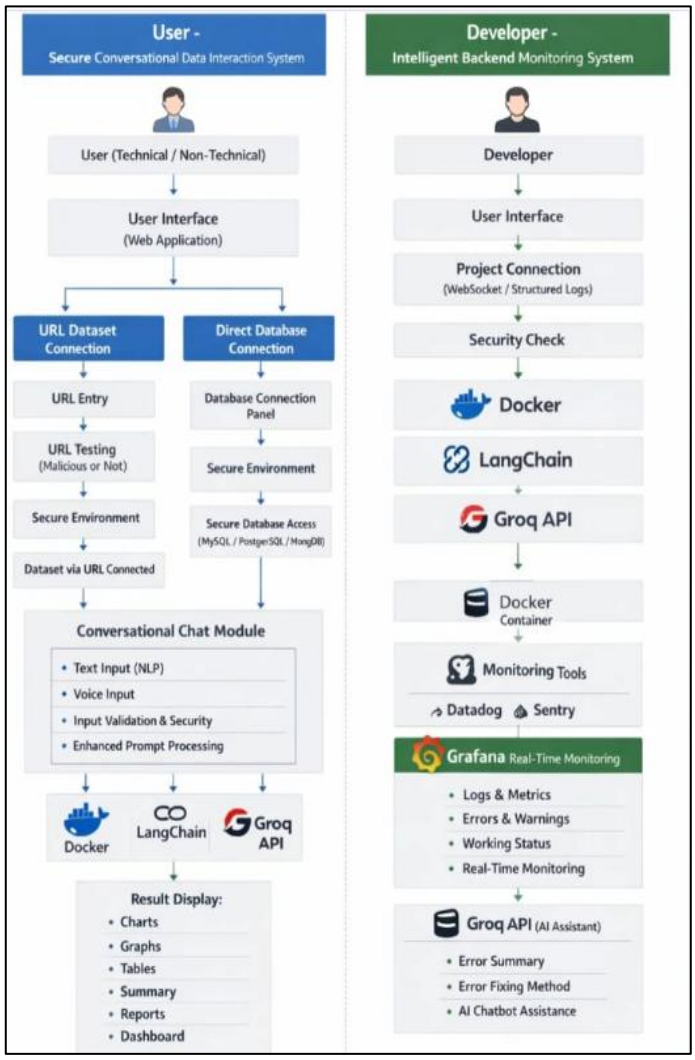

Fig -2: User Flow Diagram

5.2 Architecture of AI-Powered Conversational Assistant

The frontend uses Chainlit to provide a responsive, cross-platform conversational interface supporting contextual dialogues and real-time dashboards for database results, logs, error trends, and performance metrics.

The backend is built with FastAPI, enabling secure REST APIs and asynchronous WebSocket communication. LangChain integrates with Groq's LLM models to handle NL→SQL translation, ambiguity resolution, and AI-driven backend analysis. Queries are executed in sandboxed environments, while connected user projects stream monitoring data for live analysis.
The database layer utilizes SQLAlchemy for database-agnostic connectivity (SQLite, MySQL, PostgreSQL), storing user data and logs with encryption. Security mechanisms such as RBAC, input validation, parameterized queries, and auditing protect against injection attacks and unauthorized access.

The system supports integration with platforms like Slack and Microsoft Teams through REST APIs, and is containerized using Docker (scalable with Kubernetes) for reliability, fault tolerance, and future extensibility

5.3 Methodology

The methodology covers input processing, secure execution, and real-time monitoring. User text inputs are tokenized and analyzed via LangChain pipelines, with Groq-hosted Llama models generating optimized SQL queries or backend analyses, incorporating database schema or project context. Queries execute in sandboxed environments under RBAC, authentication, encryption, and differential privacy for aggregates.

Backend monitoring streams data from user projects connected via IDE extensions (e.g., VS Code) or WebSockets, capturing terminal output, console logs, runtime exceptions, and API events using structlog. AI agents analyze these for anomalies, providing natural language explanations and corrective suggestions. The workflow—input to processing, secure execution/analysis, visualization, and response—includes comprehensive logging for auditing. Validation was performed on a real-world diabetes dataset (public health records with patient features), demonstrating effective NL→SQL and debugging on practical data. This ensures accurate, secure, and observable interactions for reliable performance.

5.3.1 Frontend
The frontend employs Chainlit for a fast, responsive conversational UI with progressive web app support for cross-platform access (web and mobile). It handles text inputs, multi-turn queries, and contextual reasoning, displaying results, dashboards for query history, backend logs, error visualizations, and performance metrics to enable effortless understanding for all users.

5.3.2 Backend
The backend uses FastAPI for secure APIs and WebSockets, integrating LangChain with Groq API for NL→SQL translation, multi-turn context management, and backend error analysis with automated explanations. It supports query optimization, sandboxed execution, and real-time streaming from user projects for observability.

5.3.3 Database Layer
SQLAlchemy provides database-agnostic connections (e.g., SQLite for PoC, MySQL/PostgreSQL), securely storing logs, metadata, and permissions with encryption. RBAC, input validation, and sandboxing ensure traceability and prevent breaches.

5.3.4 AI & NLP Layer
This layer handles natural language understanding, multi-turn conversation state, ambiguity resolution, and intent detection via LangChain and Groq API, generating SQL or backend insights with context-aware, domain-adaptable responses.

5.3.5 Monitoring Layer
The monitoring layer utilizes structlog for structured logging, WebSockets for real-time project streaming, and embedded visualizations (e.g., Dash charts) for API flows, error timelines, and performance metrics. AI modules detect anomalies and generate explanatory responses for proactive troubleshooting.

5.3.6 Security Layer
Security employs TLS for transport, encryption at rest, RBAC via FastAPI-Users, parameterized queries, NVIDIA NeMo Guardrails for prompt rejection, and LLM Guard for scanning against injection, jailbreaks, and OWASP threats, with auditing for compliance.

5.3.7 Scalability and Integration
Docker containerization (with Kubernetes orchestration) enables scaling and fault tolerance, alongside python-dotenv for config management. REST APIs and WebSockets support integrations with collaboration tools, ensuring modular upgrades and adaptability.

6. Result

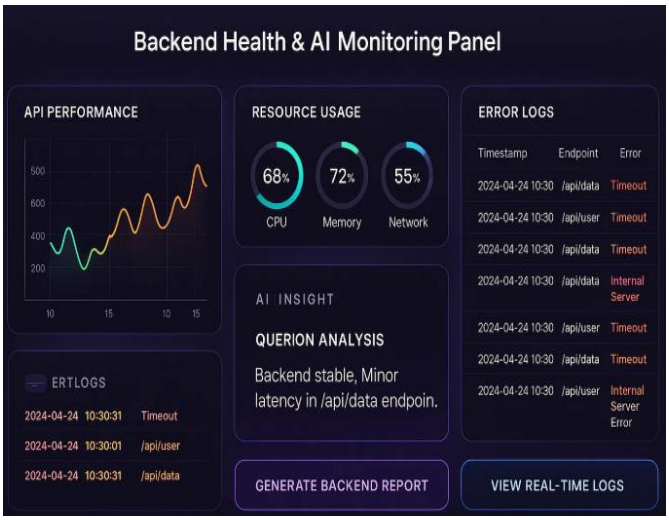

Fig -3: Backend Monitoring Dashboard

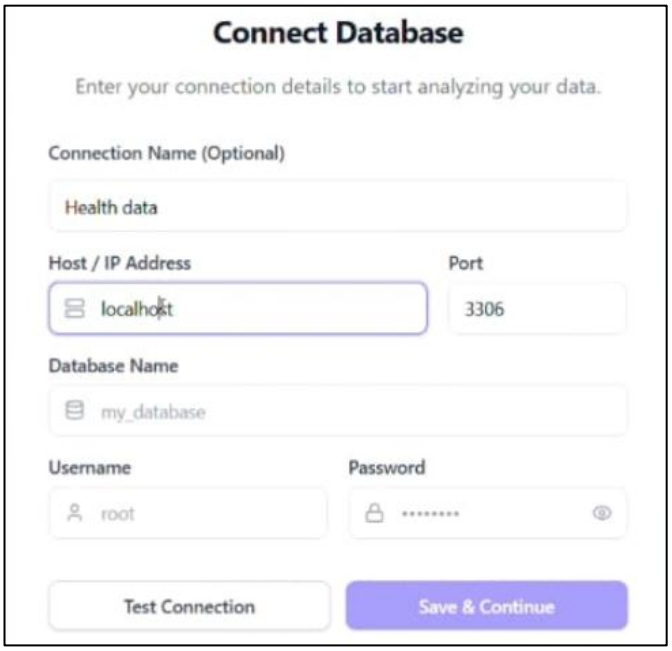

Fig -4: Database Connection

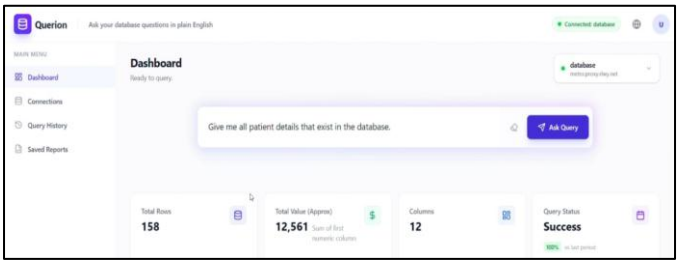

Fig -5: Conversational Database Query Interface

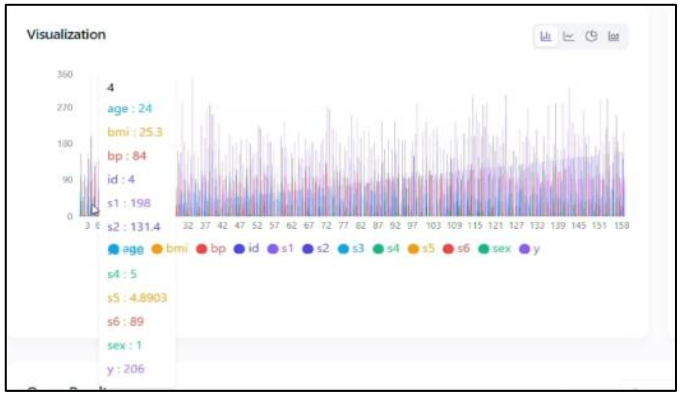

Fig -6: Conversational Database Query Interface

7. CONCLUSION

This work presented Querion, a unified platform combining LLM-based NL→SQL generation with policy-enforced query execution and real-time backend observability. Powered by Groq API and LangChain, it enables context-aware SQL synthesis and secure interaction through RBAC, parameterized execution, NVIDIA NeMo Guardrails, and LLM Guard, effectively mitigating prompt injection, jailbreaks, and OWASP LLM Top 10 threats. IDE-integrated log streaming and AI-driven runtime analysis extend the system to support intelligent debugging, error explanation, and performance monitoring, delivering transparent and developer-centric database and application management.

Future enhancements include self-hosted LLMs to eliminate API dependency, broader IDE and NoSQL support, automated code patching, predictive anomaly resolution via AI agents, and advanced visualizations with tools like Grafana. Long-term extensions may incorporate quantum-accelerated query optimization and autonomous agents for multi-database federation, positioning Querion as a scalable, future-ready solution for AI-driven data management and enterprise observability.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to the faculty and management of V. E. S. Polytechnic for providing the guidance, academic environment, and resources necessary to carry out this work. We are especially thankful to our mentor for valuable suggestions, technical direction, and continuous encouragement throughout the development of this project and preparation of this paper. We also acknowledge the classical authors and researchers whose published literature and documentation provided the theoretical and technical foundation for this study. Finally, we appreciate the support of our peers and well-wishers who contributed directly or indirectly to the successful completion of this work.

REFERENCES

[1] A. Jha, N. Anand, and H. Karthikeyan, "Conversion of natural language text to SQL queries using generative AI," in Hybrid and Advanced Technologies, CRC Press, 2025, pp. 25–32.

[2] J. Leinonen, A. Hellas, S. Sarsa, B. Reeves, P. Denny, J. Prather, and B. A. Becker, "Using large language models to enhance programming error messages," in Proceedings of the 54th ACM Technical Symposium on Computer Science Education (SIGCSE), 2023, pp. 563–569.

[3] D. Gao et al., "Text-to-SQL empowered by large language models: A benchmark evaluation," arXiv preprint arXiv:2308.15363, 2023.

[4] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers," in Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL), 2020, pp. 7567–7578.

[5] M. Fasha et al., "Mitigating the OWASP top 10 for large language models applications using intelligent agents," in 2024 2nd International Conference on Cyber Resilience (ICCR), IEEE, 2024, pp. 1–9.

[6] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2017, pp. 1285–1298.

[7] X. Liu et al., "A survey of text-to-SQL in the era of LLMs: Where are we, and where are we going?," IEEE Transactions on Knowledge and Data Engineering, 2025.

[8] Y. Luo, G. Li, J. Fan, C. Chai, and N. Tang, "Natural language to SQL: State of the art and open problems," Proceedings of the VLDB Endowment, vol. 18, no. 12, pp. 5466–5471, 2025.

[9] M. A. Ferrag et al., "From prompt injections to protocol exploits: Threats in LLM-powered AI agents workflows," arXiv preprint arXiv:2506.23260, 2025.

[10] S. Gulyamov et al., "Prompt injection attacks in large language models and AI agent systems: A comprehensive review of vulnerabilities, attack vectors, and defense mechanisms," 2025.

[11] W. Hackett et al., "Bypassing LLM guardrails: An empirical analysis of evasion attacks," in Proceedings of the First Workshop on LLM Security (LLMSEC), 2025, pp. 101–114.

[12] Y. Liu et al., "Prompt injection attack against LLM-integrated applications," arXiv preprint arXiv:2306.05499, 2023.

[13] Y. Dong et al., "Safeguarding large language models: A survey," Artificial Intelligence Review, vol. 58, no. 12, 2025.

[14] P. S. Dodd and C. V. Ravishankar, "Monitoring and debugging distributed real-time programs," Software: Practice and Experience, vol. 22, no. 10, pp. 863–877, 1992.

[15] H. Liu et al., "Real-time application monitoring and diagnosis for service hosting platforms of black boxes," in 10th IFIP/IEEE International Symposium on Integrated Network Management, 2007, pp. 216–225.

[16] B. Plattner, "Real-time execution monitoring," IEEE Transactions on Software Engineering, no. 6, pp. 756–764, 2009.

[17] F. Mueller and D. B. Whalley, "On debugging real-time applications," in ACM SIGPLAN Workshop on Language, Compiler, and Tool Support for Real-Time Systems, 1994.

[18] L. Zhang and X. Shen, "Research and development of real-time monitoring system based on WebSocket technology," in International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC), IEEE, 2013, pp. 1955–1958.

[19] G. L. Muller, "HTML5 WebSocket protocol and its application to distributed computing," arXiv preprint arXiv:1409.3367, 2014.

[20] S. Edirimannage et al., "Developers are victims too: A comprehensive analysis of the VS Code extension ecosystem," arXiv preprint arXiv:2411.07479, 2024.

[21] A. Cohen, "JavaSith: A client-side framework for analyzing potentially malicious extensions," arXiv preprint arXiv:2505.21263, 2025.

[22] I. Korontanis et al., "Real-time monitoring and analysis of edge and cloud resources," in Proceedings of the 3rd Workshop on Flexible Resource and Application Management on the Edge, 2023, pp. 13–18.

[23] U. Chindalia et al., "Real time application and CPU utilisation monitoring tool," in 2019 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA), IEEE, 2019, pp. 1136–1140.

[24] D. Orlovskyi and A. Kopp, "A business intelligence dashboard design approach to improve data analytics and decision making," in IT&I, 2020, pp. 48–59.

[25] T. Rebedea et al., "NeMo Guardrails: A toolkit for controllable and safe LLM applications with

programmable rails," in EMNLP 2023: System Demonstrations, 2023, pp. 431–445.

[26] T. Yu et al., "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," arXiv preprint arXiv:1809.08887, 2018.

[27] T. Scholak, N. Schucher, and D. Bahdanau, "PICARD: Parsing incrementally for constrained auto-regressive decoding from language models," arXiv preprint arXiv:2109.05093, 2021.

[28] R. Sun et al., "SQL-PaLM: Improved large language model adaptation for text-to-SQL," arXiv preprint arXiv:2306.00739, 2023.

[29] X. Han, S. Yuan, and M. Trabelsi, "LogGPT: Log anomaly detection via GPT," in IEEE International Conference on Big Data, 2023, pp. 1117–1122.