

**UNIVERSITY OF MYSORE**

(Re-accredited by NAAC with ‘A’ Grade)

(NIRF-2022: Ranked 33rd in University Category and 54th in Overall Category)

**MYSORE UNIVERSITY SCHOOL OF ENGINEERING**

Manasagangothri campus, Mysuru-570006

(Approved by AICTE, New Delhi)

**A Mini Project (21ADP67)**

**On**

***“*Revolutionising Database Interaction using Agentic AI.”**

Submitted in partial fulfilment for the award of the degree of

Bachelor of Engineering

In

Artificial Intelligence and Data Science

**Submitted By**

|  |  |
| --- | --- |
| **RANJAN U** | **22SEAD53** |
| **ROHITH DS** | **22SEAD56** |
| **SUDHANVA**  **H KASHYAP** | **22SEAD64** |
|  |  |

|  |  |
| --- | --- |
| Under Faculty Incharge  **Faculty name**  Asst. Professor,  Dept. of AI & DS,  MUSE, UOM, Mysuru - 570006 | Head of the Department  **Mrs Poornima K**  Asst. Professor and HOD  Dept. of AI & DS,  MUSE, UOM, Mysuru - 570006 |
|  |  |
| **Dr. M. S. Govinde Gowda**  Director  MUSE, UOM, Mysuru - 570006 | |

**D****EPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE,**

**MYSORE UNIVERSITY SCHOOL OF ENGINEERING,**

**UNIVERSITY OF MYSORE,**

**Manasagangothri campus, Mysuru-06.**

**UNIVERSITY OF MYSORE**

(Re-accredited by NAAC with ‘A’ Grade)

(NIRF-2022: Ranked 33rd in University Category and 54th in Overall Category)

**MYSORE UNIVERSITY SCHOOL OF ENGINEERING**

Manasagangothri campus, Mysuru-570006

(Approved by AICTE, New Delhi)

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**CERTIFICATE**

This is to certify that the Mini-Project (21ADP67) entitled “**Revolutionising Database Interaction using Agentic AI** ” is a bonafide work carried out by **Ranjan U, Rohith DS**, and **Sudhanva H Kashyap,** students of **VI Semester**, bearing Register No. **22SEAD53, 22SEAD56, and 22SEAD64** from the **Department of Artificial Intelligence and Data Science**, in partial fulfillment of the requirements for the award of the **Bachelor of Engineering** degree at the **Mysore University School of Engineering, University of Mysore, Mysuru.**

It is further certified that all corrections and suggestions indicated during the evaluation have been duly incorporated by the aforementioned candidate.

|  |  |
| --- | --- |
|  |  |
| Signature of the Head of the Department  **Mrs Poornima K**  Asst. Professor and HOD  Dept. of AI & DS,  MUSE, UOM, Mysuru - 570006 | Signature of the Faculty Incharge  **Faculty name**  Asst. Professor,  Dept. of AI & DS,  MUSE, UOM, Mysuru - 570006 |
|  | |
| Signature of the Director  **Dr. M. S. Govinde Gowda**  Director  MUSE, UOM, Mysuru - 570006 | |
|  | |
| **Name of the Examiners:** | **Signature with date** |
| 1. . |  |
|  |  |
|  |  |

**DECLARATION**

**We, Ranjan U, Rohith DS and Sudhanva H Kashyap,** bearing Register Nos. **22SEAD53, 22SEAD56 and 22SEAD64** of VI Semester, of **Department of Artificial Intelligence and Data Science, University of Mysore, Mysuru**, hereby declare that the **Mini-Project (21ADP67)** entitled **“Revolutionalising Database Interaction using Agentic AI”** has been duly carried out by us under the guidance of **Dr. Umera Almaz** , Asst.Professor, Department of Artificial Intelligence and Data Science, University of Mysore, Mysuru.

This Mini-Project report is submitted in partial fulfillment of the requirements for the award of the Bachelor of Engineering degree in the Department of Artificial Intelligence and Data Science by the University of Mysore, Mysuru, during the academic year **2024–2025**.

We further declare that the content of this report has not been submitted previously by anyone for the award of any degree.

|  |  |  |
| --- | --- | --- |
| Date: 11/08/2025 | **NAME** | **USN** |
| Place: Mysuru | **RANJAN U** | **22SEAD53** |
|  | **ROHITH DS** | **22SEAD56** |
|  | **SUDHANVA**  **H KASHYAP** | **22SEAD64** |
|  |  |  |

**ACKNOWLEDGEMENT**

The satisfaction that accompanies the successful completion of the **Mini-Project (21ADP67)** would be incomplete without acknowledging the blessings of the Almighty and the support of those who made this achievement possible. Their guidance and encouragement have been instrumental in the successful presentation of our project.

We are grateful to the **Department of Artificial Intelligence and Data Science, Mysore University School of Engineering, University of Mysore, Manasagangotri Campus, Mysuru**, for providing us the opportunity to enhance our knowledge through this Mini-Project.

We extend our sincere thanks to **Dr. M. S. Govinde Gowda, Director, Mysore University School of Engineering, University of Mysore**, for providing us with the necessary support, encouragement, and facilities to carry out and present our Mini-Project.

We express our heartfelt gratitude to **Mrs Poornima K, Head of the Department**, **Mysore University School of Engineering, University of Mysore**, for her constant encouragement and valuable suggestions throughout the Mini-Project.

We are deeply thankful to **Dr. Umera Almaz, Asst. Professor, AI & DS, Mysore University School of Engineering, University of Mysore**,for her guidance, motivation, and dedicated support, which played a vital role in the successful completion of our Mini-Project.

Finally, we would like to thank our family members and friends for their unwavering support and encouragement throughout this journey.

|  |  |
| --- | --- |
| **RANJAN U** | **22SEAD53** |
| **ROHTIH DS** | **22SEAD56** |
| **SUDHANVA**  **H KASHYAP** | **22SEAD64** |
|  |  |

**ABSTRACT**

The necessity of expertise in Structured Query Language (SQL) has long created a significant barrier, limiting direct database interaction to technically proficient users. This project addresses this challenge by developing an advanced agentic AI framework that revolutionizes database interaction, making it intuitive, conversational, and reliable. At its core, the system is a stateful multi-agent workflow orchestrated by LangGraph. It leverages a specialized multi-LLM strategy: Google's Gemini for initial intent analysis, Mistral's Codestral for precise SQL query generation, and DeepSeek R1 for a crucial verification and correction layer. This ensures that every query is not only syntactically valid but also semantically aligned with the user's intent. The agent employs a Retrieval-Augmented Generation (RAG) approach, dynamically fetching table schemas from a Chroma-DB vector store to provide context for the SQL generator. A key innovation is the interactive human-in-the-loop mechanism, managed through a lightweight Stream-lit interface. By successfully integrating these components, the project delivers a robust solution that democratizes data access. It transforms the complex task of database querying into a simple conversation, establishing a new paradigm for trustworthy and user-centric database interaction.

**TABLE OF CONTENT**

CERTIFICATE…….....................…………………………………………………………I - II DECLARATION…………………......................…………………………………………...III ACKNOWLEDGEMENT…………......................………………………………………….IV ABSTRACT………………………………………….......................………………………...V TABLE OF CONTENTS……………………………………….................…………..VI - VIII LIST OF FIGURES…………………………………………….....................………………IX LIST OF TABLES……………………………………………………….......................…….X

**CHAPTER 1.** INTRODUCTION.............................................................................................1

* 1. History........................................................................................................................1

1.1.1 Background and Evolution................................................................................2

1.2 Origin of the Idea........................................................................................................2

**CHAPTER 2.** PROBLEM STATEMENT................................................................................2

**CHAPTER 3.** LITERATURE SURVEY..................................................................................3

**CHAPTER 4.** OBJECTIVES....................................................................................................3

**CHAPTER 5.** SYSTEM DESIGN……………………............................................................4

5.1 HARDWARE REQUIREMENTS.............................................................................5

5.2 SOFTWAREREQUIREMENTS**...............................................................................5**

**CHAPTER 6.** IMPLEMENTATION........................................................................................5

6.1 DATA COLLECTION**...............................................................................................5**

6.2 DATA PRE-PROCESSING**.......................................................................................6**

6.3 AGENTIC PIPELINE**.................................................................................................6**

CONCLUSION AND FUTURE WORK...................................................................................7

REFERENCES...........................................................................................................................8

**LIST OF FIGURES**

Page No

Figure 1: Agent Architecture 04

Figure 2: Data Creation pipeline 06

**ACRONYMS**

|  |  |
| --- | --- |
| UI | User Interface |
| API  LSTM  LLM  SOTA  NLP  RAG  SQL  NLID  CSV | Application Programming Interface  Long Short-Term Memory  Large Language Models  State of the Art  Natural Language Processing  Retrieval-Augmented Generation  Structured Query Language  Natural Language Interfaces to Databases  Comma separated Values (Document) |

**CHAPTER 1**

**INTRODUCTION**

In an increasingly data-driven world, relational databases stand as the backbone of traditional information systems, storing vast quantities of valuable information. However, accessing this data has been gated by the need for proficiency in SQL, creating a significant barrier for non-technical users and hindering the democratization of data access . The field of NLP has long sought to bridge this gap through Agentic systems, which aim to analyse natural language questions and generate executable SQL queries. This endeavour promises to empower users from all domains to interact with complex databases as easily as having a conversation .

The evolution of Text-to-SQL has been marked by several distinct paradigms. Early efforts relied on rule-based systems and handcrafted templates, which were effective in constrained environments but lacked scalability and robustness when faced with complex database schemas or linguistic variations . The advent of deep learning brought about a significant shift, with models based on LSTM and Transformer architectures demonstrating the ability to learn the mapping between natural language and SQL syntax automatically . Then sequence-to-sequence models (Transformer era) represented a major leap forward, yet they still struggled with the nuances of complex, nested queries and cross-domain generalization. The recent emergence of LLMs has revolutionized the field, introducing a new contextual analysis with thinking power and fine-tuning capabilities. Despite their prowess, even advanced LLMs are not a panacea; they are susceptible to challenges such as generating syntactically incorrect queries, hallucinating table or column names, and misinterpreting ambiguous user intent. To address these shortcomings, contemporary research has pivoted towards more sophisticated architectures, including RAG to provide LLMs with accurate, real-time schema context, and multi-agentic frameworks that decompose the complex task into a series of manageable sub-problems solved via sequential yet cyclic agentic workflow.

## **History**

Early efforts to let humans query databases in everyday language emerged alongside relational databases in the 1970s, when SQL itself evolved from SEQUEL at IBM to operationalize Codd’s relational model. The first NLIDB were rule-based and domain-specific: linguists and system builders hand-crafted grammars, lexicons, and semantic maps from user utterances to logical forms, then to SQL, which worked in narrow domains but was brittle, costly to port, and struggled with linguistic variability. Through the 1990s and 2000s, statistical parsing began to supplement symbolic pipelines, yet systems still relied on lexicon/schema alignment heuristics and template-like mappings that relied on complex joins and nested queries. A major shift came with neural sequence-to-sequence models, initially with RNN/LSTM decoders and later Transformers, which learned to generate SQL from natural language but often produced ill-formed or non-executable queries and mis-linked question tokens to schema elements without structural awareness. To address these gaps, structure and table-aware decoders incorporated SQL grammar constraints and schema linking, improving executability and robustness on benchmarks like WikiSQL and table QA settings. Recent advances introduced fine-grained query understanding and modular pipelines (NER, entity linking, neural parsing) to better align user intent with schema and values. At present, systems increasingly combine retrieval-augmented schema/context grounding with large language models and verification steps, reflecting a maturation from handcrafted rules to hybrid, execution-aware neural and agentic architectures.

### Background and Evolution

### Text-to-SQL has evolved through four distinct eras, each addressing the limitations of its predecessors while moving closer to reliable, conversational database querying. The earliest systems (1970s–1990s) were rule-based NLIDBs built on handcrafted grammars, lexicons, and semantic parsers that mapped natural language to logical forms and then to SQL. These pipelines performed well in narrow domains but were brittle, expensive to maintain, and hard to port across schemas. They also struggled with ambiguity, paraphrase, and complex relational reasoning.

### The statistical era (2000s) introduced probabilistic parsing and machine learning for semantic role labeling, schema alignment, and lexical mapping, reducing manual engineering but retaining heavy reliance on pattern-based features and domain-specific supervision. While more adaptive, these systems still faltered on unseen schemas, multi-table joins, and nested queries.

### Neural sequence-to-sequence models (2016–2019) marked a pivotal shift: LSTMs and then Transformers learned to translate questions directly into SQL. Benchmarks like WikiSQL catalyzed rapid progress, but generic seq2seq models often hallucinated columns, produced non-executable SQL, or ignored database structure. This led to structure-aware innovations—explicit SQL grammars, syntax-constrained decoding, schema linking, and pointer networks—significantly improving executability and compositional generalization on harder datasets (e.g., Spider).

# The LLM era (2020s–present) brought powerful in-context reasoning and cross-domain generalization, yet introduced new challenges: silent failure modes, overconfident hallucinations, and sensitivity to ambiguous intent. As a result, the field moved toward hybrid and agentic designs that combine strengths: Retrieval-Augmented Generation (RAG) to ground models in real schemas and metadata; multi-step planning and tool use; and verification or self-correction loops (e.g., execution checks, critique-and-revise). Human-in-the-loop workflows further enhance reliability by resolving ambiguity at decision points.

# This project embraces that evolution: a multi-agent, RAG-grounded, verification-first pipeline orchestrated with LangGraph, using specialized LLMs for analysis (Gemini), generation (Codestral), and verification (DeepSeek R1), plus an interactive UI to keep humans in control. The result is a practical synthesis bridging linguistic flexibility with database exactness designed for trustworthy, end-to-end agentic database interaction.

## **Origin of Idea**

### The idea originated from a desire to learn and apply Agentic AI to real-world, multi-step problems with complex relational structures, while practicing end-to-end system architecture. We set out to explore how specialized agents can plan, collaborate, and verify across stages intent analysis, schema grounding, SQL generation, and validation showcasing the capabilities of modern agentic patterns. This project became a hands-on framework to evaluate the capabilities and benchmarks of the latest LLMs and agent frameworks in concert, not isolation. By integrating retrieval, multi-agent orchestration, and verification, we aimed to internalize best practices and build a trustworthy, deployable text-to-SQL pipeline.

**CHAPTER 2**

**Problem Statement**

Accessing relational data remains a persistent hurdle for non-technical members, especially when queries involve multi-table joins, nested subqueries, window functions, or temporal logic. Without an intelligent intermediary, users must translate business intent into precise SQL understanding schema nuances, naming conventions, and edge-case semantics. This forces reliance on data engineers or analysts, creating bottlenecks, context-switching overhead, and delays for routine information needs. Even for semi-technical users, evolving database schemas introduce fragility: previously working queries break silently, and intent drifts from execution. In practice, teams shoulder an ongoing burden of query maintenance, ad hoc support, and institutional knowledge transfer.

Designing a system architecture that decomposes the problem into analysable, verifiable stages demands careful orchestration. Managing state across agents for intent analysis, schema retrieval, SQL generation, verification, and execution. Cost-efficient API management is a core constraint: multi-LLM pipelines can escalate usage if not governed with caching, retries, backoff, max-token controls, and short, purpose-built prompts. Token size management is critical: schemas and prompts must be scoped and retrieved just-in-time (RAG) to avoid context bloat, truncation, or degraded model performance. Workflow design must guard against infinite loops, partial generations, and ambiguous handoffs. Absence of such a framework, organizations either accept slow, expert-mediated access or risk unreliable, unverified SQL generation both of which undermine scalable, trustworthy database interaction.

This project addresses these pain points through an agentic, Contextual analysed, verification-first pipeline with controlled API usage for cost efficiency. Resulting a SOTA framework application with several contributions to the user platform.

**CHAPTER 3**

**Literature Survey**

Text-to-SQL has progressed from early rule-based NLIDB systems to modern, hybrid pipelines that integrate large language models (LLMs), retrieval, and agentic verification, aiming to produce executable, intent-aligned queries at scale. Foundational surveys synthesize decades of progress and consistently highlight three core challenges: natural language understanding, schema grounding, and syntax- plus semantics-accurate SQL generation along with evaluation gaps and generalization across heterogeneous schemas and tasks (Deng et al., 2022; He et al., 2018; Katsogiannis-Meimarakis et al., 2022). Early neural approaches showed that generic seq2seq models (LSTM/Transformer) can learn NL-SQL mappings, but struggled with structural compositionality, schema linking, and execution robustness, motivating grammar-constrained decoding, pointer mechanisms, and schema-aware encoders (Zhang et al., 2019; Wang et al., 2022). Subsequent work broadened to systematic reviews of NL interfaces to databases (NLI4DB), emphasizing task decomposition, interaction design, and reliability in practical deployments (Gao et al., 2023).

With the rise of LLMs, surveys document a paradigm shift: prompt engineering, in-context learning, and fine-tuning significantly improve cross-domain performance, but introduce new risks including: hallucinated columns/tables, token inefficiency, and ambiguity sensitivity (Liu M. et al., 2025; Song et al., 2024). Benchmark-driven analyses show that LLM-based methods benefit from token-efficient prompt design, schema minimization, and economically constrained usage, all of which align with production constraints (Gao et al., 2023). Complementary literature underscores RAG-style grounding, combining semantic search over schemas/content with generation to eliminate hallucination and reduce context length, improving precision and scalability in real databases (Zeng et al., 2023). Industry perspectives echo these patterns, promoting agent-style orchestration, tool use, and framework for API governance for practical Text-to-SQL systems (Siddiqui et al., 2025; Liu J., 2023). A growing body of work explores multi-agent frameworks that decompose selection, generation, and refinement/verification (Song et al., 2024; Liu M. et al., 2025) often looping through execution feedback to repair erroneous SQL.

**CHAPTER 4**

**Objectives**

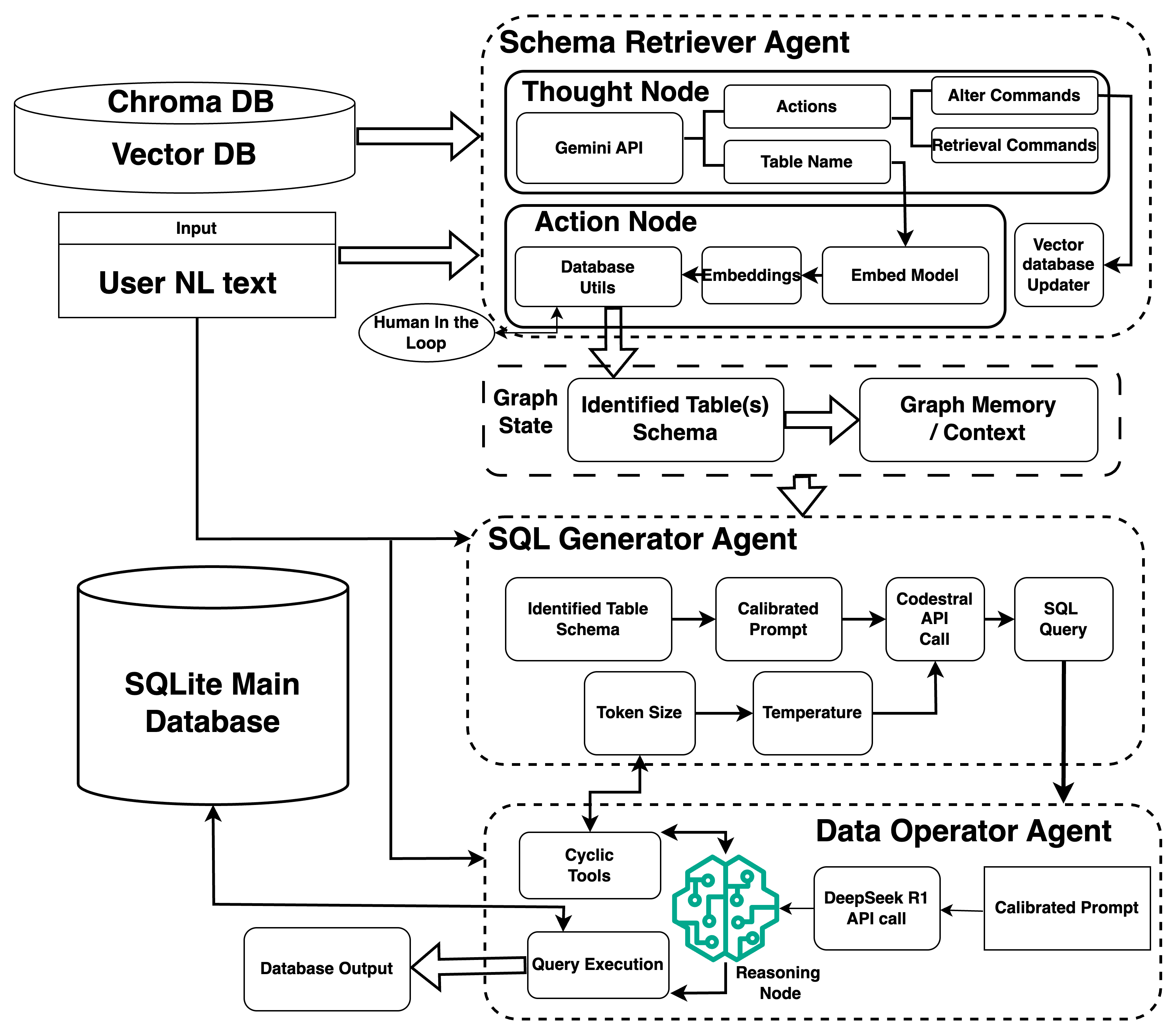
This project built upon modern frontiers, deliver a feasible solution for natural language driven database interaction and management. Our work makes several key contributions to the field, including:

1. **A Specialized Multi-Agent Framework:** Design and implementation of a stateful multi-agent system using LangGraph. The architecture moves beyond a monolithic approach by assigning distinct roles such as context analysis, query generation, and contextual validation/correction to specialized agents, resulting in a precise, modular, and maintainable workflow.
2. **A Strategic Multi-LLM Approach:** Systematic use of multiple LLMs to leverage their strengths. The system employs Google’s Gemini for nuanced intent understanding, Mistral’s Codestral for highly precise SQL query generation, and DeepSeek R1 as a dedicated verification cum database operator agent to correct, validate, and execute the generated query end-to-end.
3. **A Fully Interactive Human-in-the-Loop Interface:** Integration of human-in-the-loop mechanism via LangGraph, exposing key decision points through a user interface to enable guided intervention during the agentic workflow, ensuring clarity, control, and alignment with the desired outcome.
4. **Trustworthy Query Generation:** Inclusion of a dedicated verification and correction agent as a core trustable feature. This step ensures the final query executed on the database is not only syntactically valid but also semantically aligned with the user’s request, substantially enhancing trustworthiness and backup.

**CHAPTER 5**

**System Design.**

This system employs a reliable agentic pipeline by orchestrating specialized agents over a stateful/cyclic LangGraph, grounded by a vector database for schema recall and a SQLite database for execution illustrated in Figure 1. The design emphasizes modularity, verification-first execution, human control points, and efficient API usage.



**Fig1.** “Agent Architecture.”

Each component in the architecture is elaborated in subsequent sections.

## **5.1 Hardware Requirements**

1. **CPU:** Modern multi-core CPU (4–8 cores) recommended to handle concurrent LLM calls, vector search, and SQLite execution; more cores improve vector indexing/search throughput for ChromaDB.
2. **RAM:** Recommended minimum 6-8 GB RAM to allocate sufficient memory for in-memory indexes.
3. **Storage:** Provision 100 GB for Chroma persistence plus database and logs.
4. **GPU:** Not required until embeddings are generated using host system.
5. **Network:** Stable broadband for API calls to LLM providers and LangGraph interactions.
6. **OS:** Designed to support all OS platform hence this is a Cross-platform framework.

**5.2 Software Requirements**

1. **Python:** Python 3.10+ recommended.
2. **Agent orchestration:** LangGraph for stateful, conditional, cyclic workflows; compatible with LangChain Core for message types and tool integration.
3. **UI layer:** Streamlit for conversational interface to manage chat flow and human-in-the-loop confirmations.
4. **Vector store:** ChromaDB for schema/document embeddings with in-memory HNSW index and disk persistence. Preferred to follow performance guidance for deployment.
5. **LLM connectivity:** Gemini, Mistral Codestral, and DeepSeek/OpenRouter endpoints, environment-managed API keys and retry policies.
6. **Packaging:** Requirements stated in requirements.txt that adhere to LangGraph deployment guidance for version compatibility.

**CHAPTER 6**

**Implementation.**

This system implements Text-to-SQL pipeline using a multi & sequential agent architecture driven by LangGraph, grounded by RAG over a Chroma vector store, and safeguarded by DeepSeek R1 verification. It ingests scraped exam results, builds SQLite database and chroma vector databases, generates schema-aware SQL, verifies correctness, executes, and presents results via Stream-lit UI.

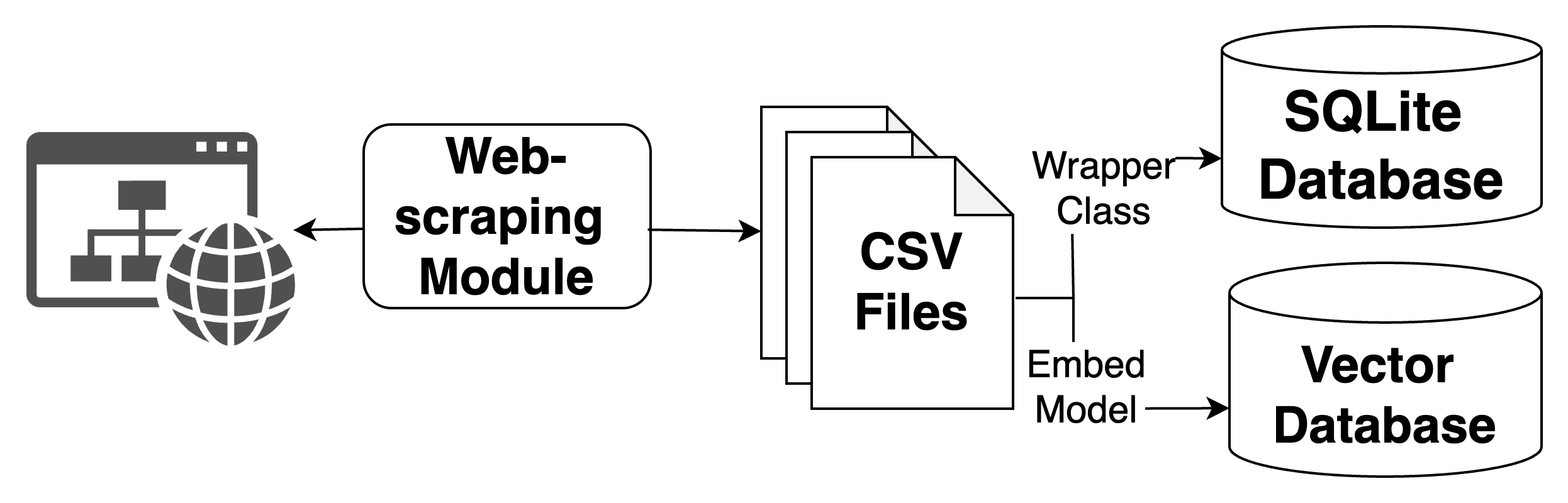
**6.1 Data Collection.**

Data collection was conducted via web scraping of university results portals to assemble per-candidate records across exam batches. The scraper, implemented uses Selenium WebDriver with headless. It programmatically navigates to the results site, fills required inputs (registration number and a valid default date of birth), submits the form, and extracts name, semester, and GPA fields across multiple exam batches.

For dynamic elements such as result summary, dropdowns, etc. Re-location of stale elements before interacting with dropdowns, and fallback logic are used. The scraper iterates sequential registration numbers. GPAs are parsed with a simple converter and aggregated per student with an average computed over available batches. The output is persisted to CSVs by cohort and branch (AIDS/CSD/AIML) for downstream processing. This approach balances volume with throughput, resulting in producing clean, uniform CSVs ready for database construction.

**6.2 Data Pre-processing.**

The pre-processing stage comprises two artifacts: (1) construction of the SQLite database and (2) creation of a Chroma vector database for schema retrieval which is depicted in Figure 2.



**Fig 2. “Data Creation pipeline”**

SQLite database creation transforms the scraped CSVs into normalized tables with coherent naming and consistent keys. Each batch/branch file is loaded into a dedicated table and, where present, the regno column is set as the primary key to support unambiguous joins and fast lookups. The loader script walks the results directory tree, reads each CSV via pandas, and writes tables into database. This phase of data pre-processing step results in creation of SQLite database with 9 tables. Each branch with 3 batches (1st, 2nd, 3rd consecutive batches) are considered to be separate tables.

Chroma vector store is constructed to power retrieval-augmented schema grounding. The script connects to the SQLite database, extracts table schema from each table of the database and synthesizes compact documents of the form “Table: <name> Schema: <SQL>,” plus a global overview listing all objects. Embeddings of these extracted table schema is computed using SentenceTransformer, then persisted into a chroma PersistentClient collection. This pipeline also manages versioned backups of the persist directory to keep previous indices safe, and ensures atomic insertion with batched adds.

This RAG layer allows the agent to retrieve only the relevant table schema(s) at generation time, minimizing token usage, reducing hallucination risk, and increasing SQL Query precision. Together, the SQLite store and the Chroma vector database form a clean, queryable substrate that the agentic layer can reliably leverage.

**6.3 Agentic Pipeline.**

The aim of the implemented agentic pipeline is to provide a modular, and efficient path from a user’s natural language request to an executed SQL query with verifiable intent alignment. It decomposes the problem into specialized roles—analysis, schema retrieval, generation, verification, and execution while preserving state and enabling human intervention at ambiguity points. The framework uses LangGraph, which structures the workflow as a directed, conditional graph with clearly defined nodes, transitions, and stopping conditions, avoiding infinite loops and redundant API calls.

This implementation follows the programming structure called SOLIDS. This structure is chosen in vision of better code maintainability, easier deployment, uniformity across framework.

The pipeline begins with intent analysis (thought node) powered by Gemini. Here, the system ingests the user’s NL text and infers the table(s) involved, extracted entities, and the SQL command type. With prompt engineering calibrated to the selected database, this step encapsulates early disambiguation: When entities are present but the table remains ambiguous, a semantic search node queries the Chroma vector store by content and entity, if the table is uncertain, the agent marks a need for clarification.

Human-in-the-loop interaction is managed entirely in the Stream-lit interface. Once confirmed, the action node retrieves the exact schema text for the chosen table(s) from the Chroma vector database. This retrieval is passed through LangGraph’s memory module to optimise the rate of API calls per query. This memory is utilised by overall agent workflow in future for subsequent processes. This memory with contextual hints including: table schema, detected command type and relevant entities, guide the query generator.

SQL generation is handled by a Mistral’s Codestral-backed node that prompts the model with the minimal necessary schema context and user intent, producing a single, complete SQLite-compatible query. The generator is configured with token controls, determinism-leaning temperature, and a structure-aware prompt to encourage unions across different table. The prompt is well calibrated to suit the decided workflow hence the token size for every API call can be varied by subsequent agent. This effective functionality in the workflow optimises the process with cost effectiveness. Since unnecessary tokens drains the units in API calls resulting in draining resources.

The generated SQL query then flows to the Data-Operator agent, powered by DeepSeek R1, which verifies that the query is both syntactically executable in SQLite and semantically aligned with the NL intent. This agent serves three roles: correction of minor syntax issues, detection of truncation/incompleteness (triggering controlled regeneration with adjusted token budgets), and final execution on the SQLite database if the query is deemed perfect or reliably corrected. Results are then returned as structured rows and displayed in the interface, alongside the finalized SQL and verification narrative.

State management across nodes stores messages, schema, entities, generated SQL, retry counts, and UI flags for confirmation, ensuring idempotent transitions and predictable behaviour. API usage is governed by caching, retry/backoff policies, and careful prompt scoping to maintain cost efficiency. Similar querying the agent results in repeated API calls which is undesirable. Therefore, the framework employs caching of query, and data states which can be fetched immediately when the agent is queried repeatedly with same query. The Stream-lit UI centralizes the entire interaction loop including: query input, table confirmation, SQL visibility, verification reasoning, and result inspection.

This agentic pipeline operationalizes best practices in modern Agentic workflow: retrieval-enhanced grounding, multi-LLM specialization, explicit verification before execution, and human oversight at ambiguity points. The result is a practical, maintainable system that elevates database interaction and management from command syntax to conversational intent while preserving the guarantees professionals expect from production querying.

**Conclusion and Future Work**

This project delivers a practical, verification-first agentic system that transforms natural language into safe, executable SQL through an agentic, retrieval-grounded, and human-centric sequential **/** cyclic workflow. By orchestrating specialized agents: Gemini for intent analysis, Codestral for schema-aware SQL generation, and DeepSeek R1 for verification and execution over a Chroma-backed RAG layer, the system achieves reliability, transparency, and usability. The Stream-lit user interface places users in control, enabling ambiguity resolution, exposing finalized SQL, and surfacing reasoning. Architecturally, the design balances precision and cost through scoped prompts, node memory, caching, retries, and controlled token budgets; operationally, it scales from simple lookups to multi-table patterns while preserving trust through explicit checks and human-in-the-loop.

**References**

1. Deng, N., Chen, Y., & Zhang, Y. (2022). Recent Advances in Text-to-SQL: A Survey of What We Have and What We Expect. *Proceedings of the 29th International Conference on Computational Linguistics*.
2. Gao, D., Chen, W., & Wang, T. (2023). Text-to-SQL Empowered by Large Language Models: A Survey. *Proceedings of the VLDB Endowment, 17*.
3. He, Y., Bai, D., & Jiang, W. (2018). *Text-to-SQL Translation with Various Neural Networks*. Stanford University.
4. Katsogiannis-Meimarakis, G., & Kermanidis, K. L. (2022). A survey on deep learning approaches for text-to-SQL. *The VLDB Journal, 32*(1), 103-128.
5. Liu, M., & Xu, J. (2025). *NLI4DB: A Systematic Review of Natural Language Interfaces for Databases*. arXiv preprint arXiv:2503.02435.
6. Liu, J. (2023). *Combining Text-to-SQL with Semantic Search for Retrieval Augmented Generation*. LlamaIndex Blog.
7. Pourreza, M., & Rafiei, D. (2023). A Survey on Employing Large Language Models for Text-to-SQL. *ACM SIGMOD Record, 52*(4), 25-36.
8. Siddiqui, A., Srivastava, P., & Vadupu Lakshman Manikya, P. K. (2025). *Building a custom text-to-SQL agent using Amazon Bedrock and Converse API*. AWS Machine Learning Blog.
9. Song, L., et al. (2024). A Survey of LLM-based Text-to-SQL. *arXiv preprint arXiv:2406.08426*.
10. Wang, J., Ng, P., Li, A. H., Jiang, J., Wang, Z., Nallapati, R., Xiang, B., & Sengupta, S. (2022). Improving text-to-SQL semantic parsing with fine-grained query understanding. *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*.
11. Zeng, Z., et al. (2023). A Multi-Agent Collaborative Framework for Text-to-SQL. *arXiv preprint arXiv:2312.11242*.
12. Zhang, Z., et al. (2019). *Text to SQL Query Conversion Using Deep Learning*. SSRN.