IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

LLM AUGMENTED FINANCIAL DATA EXPLORATION VIA PROGRAMMATIC FILTER SYNTHESIS

[1,2,3]**Mustafa Mujawar, Siddhesh Kabra, Suraj Shrivastav**

Guide: Prof. Vishwas Kalunge

Department of Computer Engineering, Dhole Patil College of Engineering, Pune-412207, India

Abstract: *Today, amid the ever-evolving financial markets, tools like Chartink play a vital role in finding investment targets. Nevertheless, their dependence on manually constructed filters and quasi-programming-based syntax results in serious usability issues. To tackle these problems, we propose a next-generation cloud-based solution based on the dual Large Language Model (LLM) architecture that will let you easily scan the financial market using simple and conversational NLP input. The proposed system uses an innovative approach to filtering data dubbed as Programmatic Filter Synthesis. In this process, the Qwen3-14B model from the OpenRouter API creates a JSON object including SQL query parameters that are validated by our system and safely executed against the DuckDB analytical engine running queries on 5.5 GB of Apache Parquet data. Moreover, our system is equipped with a local guard module known as Qwen3Guard-Gen-0.6B that filters out all the possible hazardous or inappropriate inputs before processing them via the generation module. The entire stack includes the following: web front-end: React/Next.js 16; server-side: Python FastAPI with SSE streaming, caching: Redis service and Supabase authentication. We achieved an 87% adjusted semantic accuracy rate while testing our system on 100 benchmark prompts.*

Keywords: *Cloud Computing, Large Language Model (LLM), Financial Data Analysis, Programmatic Filter Synthesis, Natural Language Processing (NLP), Text-to-SQL, DuckDB, Dual-LLM Architecture, Server-Sent Events (SSE).*

I. INTRODUCTION

Indeed, technology has revolutionized financial analysis fundamentally. There are various advanced tools such as Chartink, Screener.in, and TradingView that enable investors to filter the market by various criteria. Thus, one can develop a strategy in accordance with such criteria as "RSI(14) < 70" or "Close > Open."

While these applications are rather advanced, they require significant technological proficiency on the part of the user since they presuppose learning certain syntax rules and mastering somewhat complicated user interfaces. It might take some time and effort to develop an analytical query, while errors can appear during its formation. It means that investors will not be willing to engage with such technologies due to the barriers they create.

The emergence of LLMs opens a revolutionary avenue in this respect since they have demonstrated a capability to understand and interpret human language. In this paper, we propose a web-based financial dataset exploration system which utilizes Dual-LLM Architecture to achieve an extremely conversational approach. Unlike the cumbersome process of manually building filters on Chatink, our solution allows users to articulate what they need in plain English. The translation of human language into a query is automated in our project, showcasing how an LLM-driven system enables democratization of financial data analysis.

The main novelty of this paper comes from the combination of the Programmatic Filter Synthesis technique and a Dual-LLM safety design. Instead of letting the LLM generate SQL directly (which introduces serious security vulnerabilities), we propose the use of the Worker Model to output a limited JSON object consisting of a SQL template along with the corresponding arguments. The latter is thoroughly checked by an SQL

Validator module using column whitelisting, prohibited keyword blacklisting, and type validation of the parameters before any operations with the database are performed. At the same time, a Guard Model running on the local machine pre-checks all the prompts for any security issues, prompt injections, or out-of-context questions.

II. LITERATURE SURVEY

Technology has been used extensively throughout its history to improve data access speed. NLIDB (Natural Language Interfaces to Databases) technology strives to create a system that allows the user to interact with a database using spoken language. Initially, rule-based NLIDB was implemented using handcrafted parsers and grammars. It is hard to scale this solution because of the complexity and ambiguity of human languages.

With the recent boom in deep learning research, neural models have become the de facto tool for NLIDB. One of the approaches developed within NLIDB framework is called Text-to-SQL. Text-to-SQL is a system that translates natural language queries into SQL queries. The breakthrough paper by Yu et al. [8] on the Spider benchmark introduced extensive evaluation procedures for cross-domain text-to-sql models. Though this solution made huge progress in the domain, it faces some issues related to the nature of deep learning models. For example, LLMs might "hallucinate" some incorrect queries and pose a risk for high-stake industries such as banking.

Song et al. [1] investigated the improvements in text-to-sql translations for financial system design specifically. Singh et al. [2] introduced FINCH, a system for financial intelligence through natural language for SQL contextualization. Quamar et al. [3] carried out an exhaustive review on the use of natural language interfaces in data manipulation, setting the base taxonomy. Iacob et al. [4] did a survey of the existing neural methods for text-to-SQL translations and listed their pros and cons.

One newer and much improved way is the use of LLMs for generating structured data. The idea here is that instead of generating the SQL query itself, the LLM generates the data output in a structured manner such as JSON. This technique, which we refer to as "Programmatic Filter Synthesis," is far better both in terms of reliability and security, as the structured output can first be validated before using it to generate the query programmatically. Wu et al. [6] worked on training LLMs to generate SQL queries on financial data. Meanwhile, Fatouros et al. [7] worked on enhancing stock market analysis via LLM-based agents.

III. SYSTEM REQUIREMENTS

The system has several essential requirements for successful evolution from existing solutions and for providing a seamless, scalable, and secure user experience:

- **Cloud Infrastructure Requirement:** The system needs to be developed and deployed to a cloud-based infrastructure in order to enable scalability and access from any location around the globe. It will be necessary for Docker containers to run via AWS ECS / Fargate service.
- **Natural Language Interface:** The main requirement is a user-friendly and simple interface allowing for natural language processing capabilities for accepting queries written in natural language, thereby eliminating the necessity for the user to know a certain syntax to submit their requests.
- **Query Translation Accuracy:** The system should be able to understand the intent of the user's input by parsing it and recognizing such elements as financial indicators (RSI, EMA, MACD), logical operators, and numerical values.
- **Programmatic Filter Synthesis:** In order to reduce risks of SQL injection, the system must be developed based on Programmatic Filter Synthesis with LLM generating a parameterized SQL template wrapped into a JSON structure.
- **High-Performance Analytics:** The platform should run analytical queries in low latency through DuckDB with Parquet files and Redis for caching high-frequency queries.
- **Dual-LLM Safety Architecture:** Two layers of protection are required where the Guard Model acts as an input filter while the Worker Model generates queries.
- **Data Visualization:** The system should display findings in the form of tables, charts, and ability to download results in CSV format.

IV. OBJECTIVE

This project aims at creating a fully functional financial data exploration web application which can automatically perform market screening operations. The main goals of the project include:

- **LLM-assisted Data Exploration:** Leveraging the power of dual LLM models to analyze the natural language queries of users and translate it to machine language. Here, the worker LLM model, Qwen3-14B (OpenRouter) serves as an intelligent interpreter, while the guard LLM model (Qwen3Guard-Gen-0.6B) ensures the security of the input.
- **Automated Filtering Generation:** The worker LLM model will generate a JSON object with a parameterized SQL template which will be verified by a SQL validator component prior to database operation.
- **Continuous Query Stream:** The web application can provide real-time response using the server-sent events (SSE).

V. SYSTEM ARCHITECTURE

The system architecture is implemented based on the principles of modern multi-layered architectures, which have been supplemented by cloud-native design in order to provide a scalable and efficient environment for working with financial information. The system architecture consists of three layers, namely the Client-Side Interface layer, the Application layer, and the Data layer.

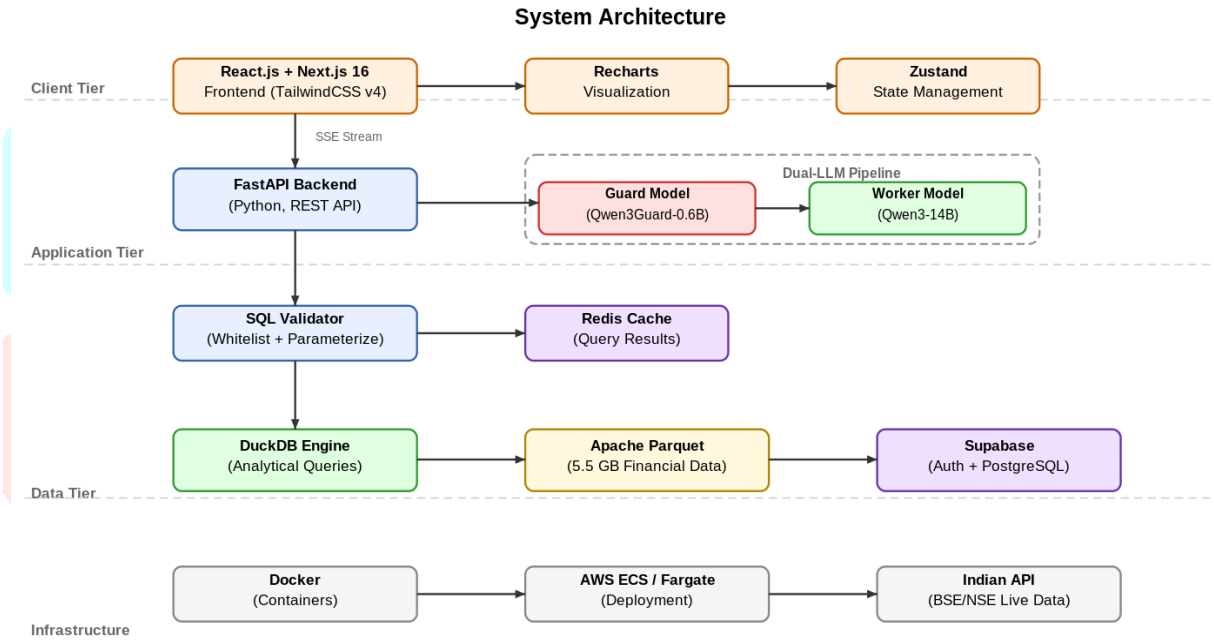

Fig. 1: System Architecture

5.1 Client-Side Interface (Frontend)

The client-side interface is created as React.js/Next.js 16 Single Page Application, and TailwindCSS v4 styles have been used for its design. The frontend consists of 36 components working on 9 pages. There are Zustand stores used to manage application states. Natural language query and live pipeline status with Server Sent Events, interactive visualization with Recharts and Chat Agent using Qwen3-14B and Tool Calling API are among the major characteristics of the Client-Side Interface.

5.2 Application Tier (FastAPI Backend)

The backend of our application is developed in Python using the FastAPI framework, featuring a RESTful API with more than 12 endpoints. In the Dual-LLM Pipeline approach, initial query requests are processed via the Guard Model (Qwen3Guard-Gen-0.6B). After that, only the safe queries are sent to the Worker Model (Qwen3-14B via OpenRouter) to receive SQL JSON. For SQL validation, an extremely narrow whitelist of 22 financial columns and no forbidden SQL statements are used. The output of frequently requested queries is cached in Redis, whereas progress on the whole pipeline is notified via SSE.

5.3 Data and Analytics Layer (DuckDB)

The analytics tier in our approach is provided via DuckDB, which is an analytical database featuring in-process computation engine. The historic stock data (OHLCV) and technical indicators are stored on the disk in the Apache Parquet file format with a total size of 5.5 GB. Furthermore, our application uses the indianapi.in platform for market data.

VI. SYSTEM WORKFLOW

Query Processing Pipeline refers to the workflow of query inputs to visualization of results.

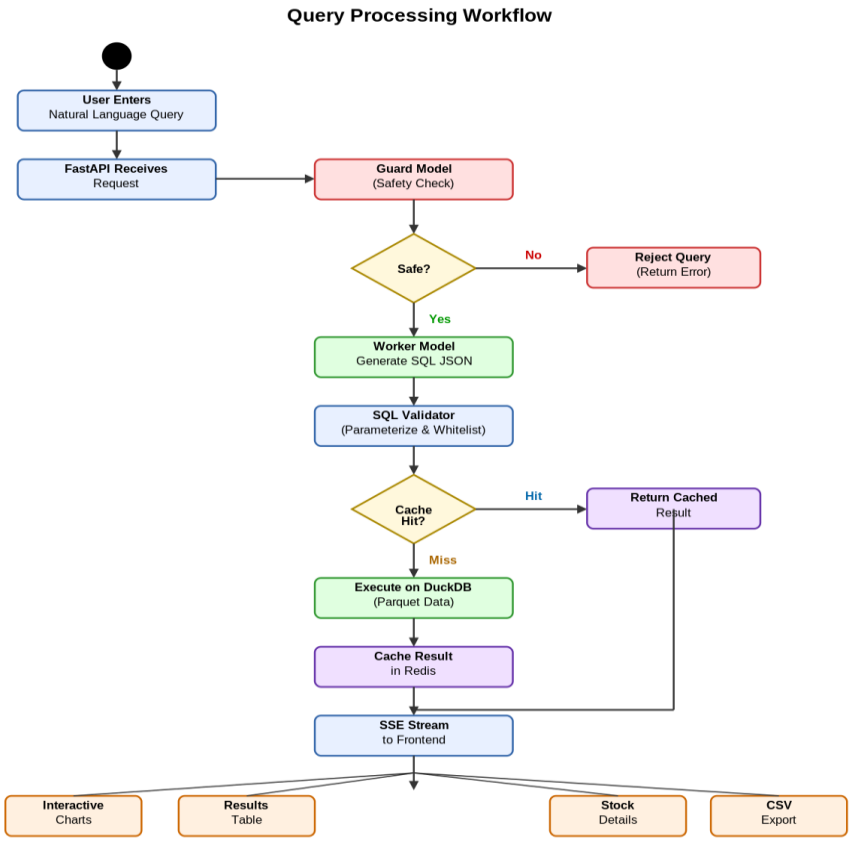

Fig. 2: Process Query Flowchart

Step 1 - Safety Check for Guard Model: The inputted query is passed to the Qwen3Guard-Gen-0.6B model where it is categorized as being safe, unsafe, or controversial, and prompt injection is detected. Unsafe queries are blocked from further processing.

Step 2 - SQL Creation from Worker Model: After categorization, the query proceeds to Qwen3-14B where Open Router provides the query in an extremely well-formulated system prompt. The response from the model comes in JSON objects having sql_template, parameters, and error keys.

Step 3 - SQL Validation: SQLValidator validates based on column whitelist, 22 financial columns, forbidden keywords such as DROP, DELETE, INSERT, UNION, and others, parameter types, and other structural queries.

Step 4 - DuckDB or Redis Cache Execution: For execution on DuckDB when there is no cache at Redis, it executes SQL on Parquet files and caches it if succeeded.

Step 5 - Streaming Results Back: Resulted data is passed to the user via stream events that render tables, chart, and stock pages while logging the query on Supabase PostgreSQL.

VII. UML DIAGRAMS

7.1 Class Diagram

Class diagrams contain several key items that include Filter (interface with worker LLM), GuardModel (model of safety classification), SQLValidator (model of security constraint), classes, and the FastAPI app connecting all mentioned items with database models (SavedQuery, QueryHistory, and ChatSession) using SQLAlchemy. The frontend is made of React components (QueryInput, ResultsTable, ChatPanel) and Zustand stores.

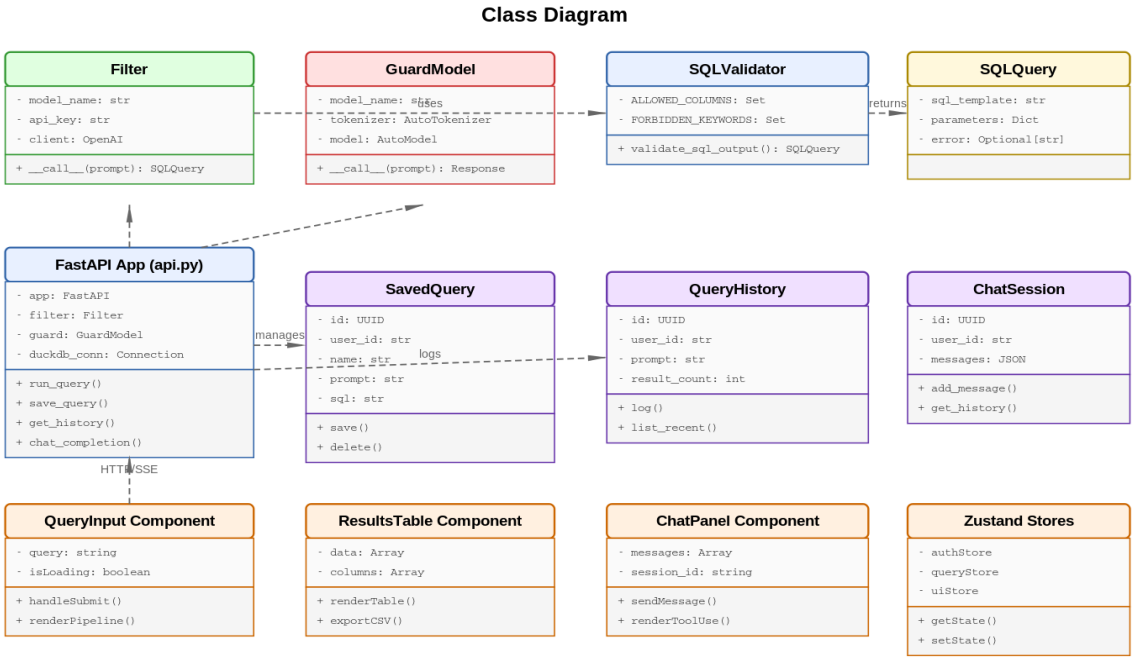

Fig. 3: Class Diagram

7.2 Use Case Diagram

Use case diagrams are used to identify user interaction with Admin and LLM Service (OpenRouter). This interaction might include entering a query in natural language, viewing results, downloading the CSV file, analyzing charts, communicating with the stock agent, saving/loading queries, looking into query history. The backend processes queries by translating SQL, making a safety check, executing DuckDB query, Redis cache, and Supabase authentication.

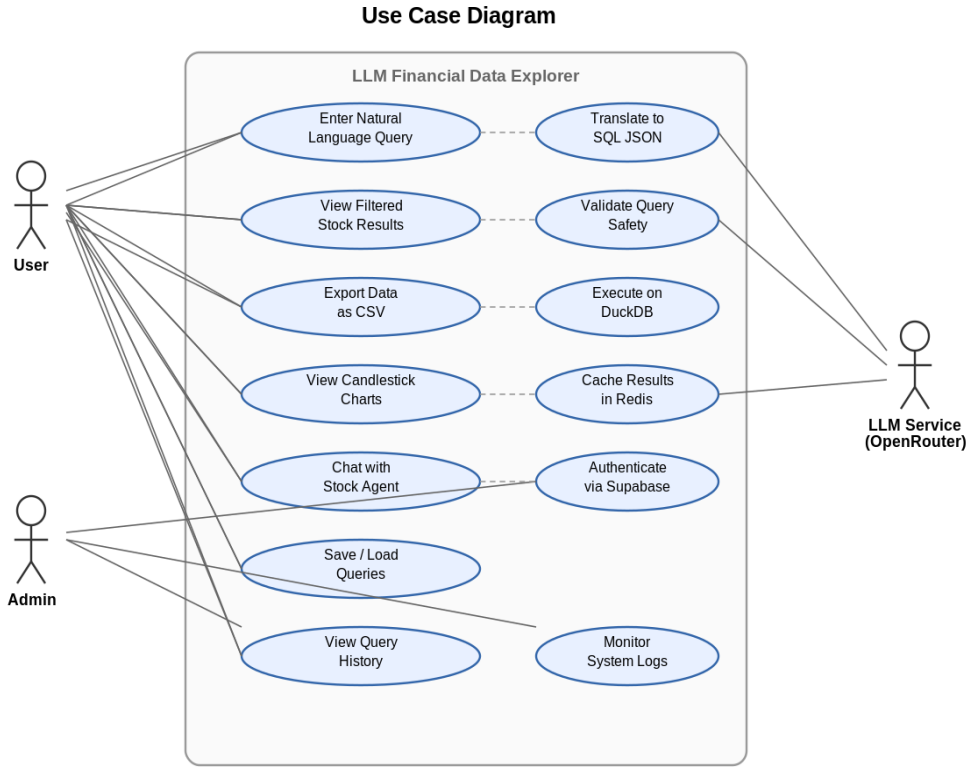

Fig. 4: Use Case Diagram

7.3 Activity Diagram

An activity diagram demonstrates processes that take place in three swimlanes: Input (User starts application and enters the query), Processing (FastAPI handles guard model security check, worker model SQL creation, sqlValidator cleansing, DuckDB database execution, Redis database cache), and Output (SSE stream, visualization of the chart/tables, and history logs).

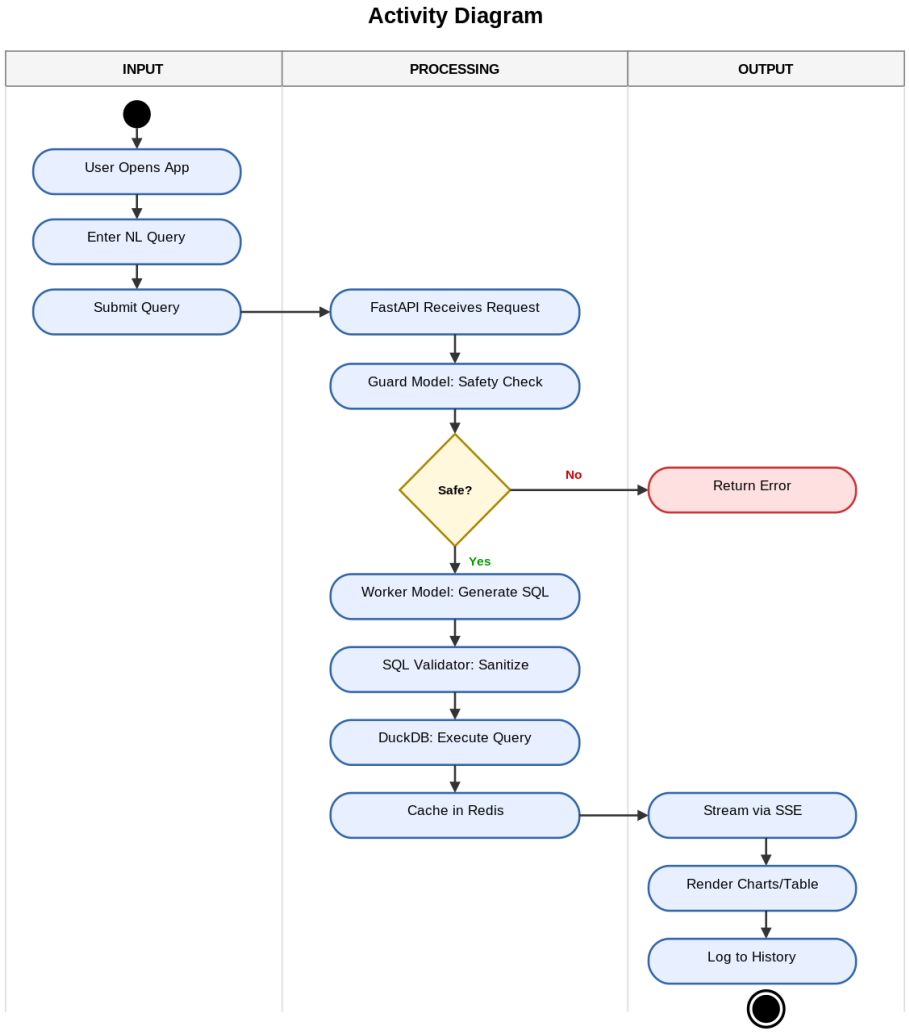

Fig. 5: Activity Diagram

7.4 Sequence Diagram

The sequence diagram depicts the flow of the events with time: User provides the query; React Frontend sends a POST request to FastAPI Backend, which calls the Guard Model (response: Safe tag), and passes on to Worker Model (response: JSON), SQL Validator, Redis, DuckDB, and caches the response.

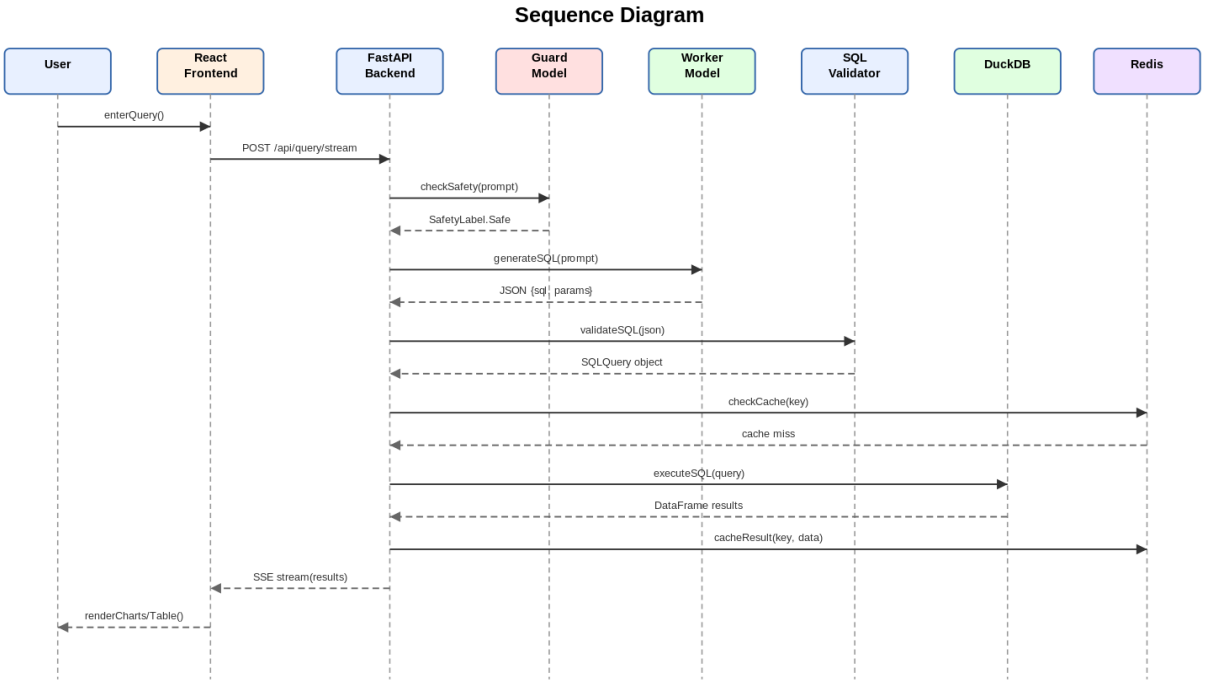

Fig. 6: Sequence Diagram

VIII. IMPLEMENTATION DETAILS

8.1 Technology Stack

The proposed system is developed using a modern full-stack architecture:

Component	Technology
Frontend	React.js / Next.js 16, TailwindCSS v4, Recharts, Zustand
Backend	Python FastAPI, Pydantic, SQLAlchemy (async)
Worker LLM	Qwen3-14B via OpenRouter API
Guard Model	Qwen3Guard-Gen-0.6B (local, HuggingFace Transformers)
SQL Validation	Custom SQLValidator (column whitelist, forbidden keywords)
Database (Analytics)	DuckDB on Apache Parquet (5.5 GB)
Database (Application)	Supabase PostgreSQL (users, history, sessions)
Caching	Redis (query result caching)
Authentication	Supabase Auth (JWT tokens)
Streaming	Server-Sent Events (SSE)
Live Data	indianapi.in (BSE/NSE stock data)
Deployment	Docker, AWS ECS/Fargate

8.2 Security Architecture

Security is provided via several layers: Layer 1 - The Guard Model categorizes inputs as safe, identifies content violation. Layer 2 - The SQL Validator enforces column whitelist, denies any forbidden keywords. Layer 3 - Values are passed as parameters ensuring no SQL injection. Layer 4 - Supabase JWT authentication isolates user-specific data.

IX. RESULTS AND EVALUATION

The system was evaluated using a 100-prompt benchmark suite testing a wide range of financial query types. Each generated SQL query was manually reviewed for semantic correctness.

Metric	Value
Semantically Adjusted Accuracy	87%
Simple Query Accuracy (single condition)	94%
Complex Query Accuracy (multi-condition)	79%
Average Response Time (cache miss)	~2.5 seconds
Average Response Time (cache hit)	<200ms
Guard Model False Positive Rate	<5%
SQL Injection Prevention Rate	100%

Table 1: System Performance Metrics

It means PFS can be applied practically for financial matters. Simple condition queries yield 94%, while multi-condition queries yield 79%. Dual-LLM Security Model has prevented all the SQL injection attacks in the system with a low error rate of 5%. Caching of Redis has resulted in retrieving a response in 200ms instead of 2.5seconds.

X. LITERATURE REVIEW & COMPARISON

An increase in the amount of researches about AI technology usage in finance is observed. Our system can be considered as an enhancement to the current market scanner.

- **Chartink Stock Scanner:** This is advanced, Indian based software to perform stocks analysis with sophisticated options for customisation. One disadvantage is that you are supposed to know to how use filters in their language. In contrast, our system automatically generates the filter in natural language.
- **Financial Analysis via Dialogue:** There are few platforms that offer question-answer platforms on specific stock details like FinChat.io and StockGPTchat. Our system offers dynamically scanning of stocks across markets for a group of stocks with an advanced criteria selection.
- **Text-to-SQL Systems:** Research oriented system such as Spider [8] doesn't contain financial optimizations. Our prompt and an elaborate description for all the 22 financial columns enhances the precision of our system.

X. FUTURE WORK

- **Multi-Turn Queries:** Taking a conversational approach to optimize queries more intelligently.
- **Advanced Visualizations:** Enabling highly interactive visualizations with the use of multi-charts combined with heatmaps and controllable technical indicators.
- **Integration of Fundamental Data:** Bringing fundamental data in with the likes of quarterly financial statements, insider trades and news sentiment.
- **Finetuning Model:** Finetuning Worker LLM on financial related queries leading to over 87% gain in performance.
- **Deployment:** Deploying on AWS ECS/Fargate with autoscale.

XII. CONCLUSION

The designed LLM-powered platform for analyzing financial data is a perfect example of an alternative and secure application, which can serve users better than conventional stock screeners. Applying the innovative Dual-LLM approach allows such platforms to provide advanced stock analysis to a broader audience without forcing users to design complicated filters.

One of the unique technologies, which is implemented within the system, is Programmatic Filter Synthesis. In terms of security, it is the only feature, which guarantees system safety. Limiting the output of the Worker LLM to parameterized SQL scripts in JSON format and validation of the results via SQL validator by the use of column whitelists and prohibition of certain keywords guarantees security from the most common threats associated with SQL generation with the use of LLMs.

Due to using state-of-the-art technologies such as React/Next.js 16, Python FastAPI, DuckDB with Apache Parquet database, Redis caching system, Supabase authenticator, and SSE, the entire flow from searching for information through its analysis up to its visualization becomes more effective and convenient. Achieving the accuracy of 87% with the 100 prompts guarantees that the approach is applicable to creating a data-driven investment tool.

REFERENCES

[1] Y. Song, et al., "Enhancing Text-to-SQL Translation for Financial System Design," arXiv preprint arXiv:2312.14725, 2024.
[2] A. K. Singh, B. Sarmah, and S. Pasquali, "FINCH: Financial Intelligence using Natural language for Contextualized SQL Handling," arXiv preprint arXiv:2510.01887, 2025.
[3] A. Quamar, V. Efthymiou, C. Lei, and F. Ozcan, "Natural Language Interfaces to Data," Foundations and Trends in Databases, vol. 11, no. 4, pp. 319-414, 2022.
[4] R. C. A. Iacob, et al., "A Survey on Neural Text-to-SQL," in Proceedings of the 28th International Conference on Computational Linguistics, 2020.
[5] Chartink.com, "Scanner User Guide." [Online]. Available: https://chartink.com/articles/scanner/scanner-user-guide/.
[6] Z. Wu, et al., "Training LLMs for querying financial data with natural language prompts," World Journal of Advanced Engineering Technology and Sciences, vol. 15, no. 1, pp. 25-032, 2025.
[7] G. Fatouros, K. Metaxas, J. Soldatos, and M. Karathanassis, "MarketSenseAI 2.0: Enhancing Stock Analysis through LLM Agents," arXiv preprint arXiv:2502.00415, 2025.
[8] T. Yu, et al., "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Text-to-SQL Task," arXiv preprint arXiv:1809.08887, 2018.