Stock Roulette: LLM edition

A Distributed, Data-intensive Web application

**DATA 608 – Summer 2025**

**Project report**

Franco Doi

Gautham Nagaraj

# 1. Problem statement and concise summary of the results

## 1.1 Problem statement

The rapid convergence of **large language models (LLMs)**, **real-time data processing**, and **interactive web-scale systems** has transformed how intelligent applications are designed and deployed. In finance, the lack of accessible, interactive training tools using authentic market data leaves aspiring traders with limited options - often forcing them to enter real markets without adequate preparation.

**Stock Roulette: LLM Edition** addresses this gap by offering a distributed, data-intensive web application that simulates one month of historical stock trading in a ~5-minute, fast-paced game. The system integrates a cloud-hosted LLM-powered advisor to provide real-time, context-aware trading recommendations.

Beyond entertainment, Stock Roulette: LLM Edition serves as an educational platform. It allows students, hobbyist traders, and training participants to explore market dynamics, experiment with AI-assisted decision-making, and develop critical data literacy skills in a safe, gamified environment. Instructors can integrate the tool into coursework to demonstrate concepts in time-series analysis, feature engineering, and real-time data systems.

The application is underpinned by a modular data engineering pipeline, designed for scalability, reproducibility, and low-latency performance under concurrent user loads.

## 1.2 Summary of Results

The project successfully delivered:

* **Functional Gameplay**Each session assigns a random month/year and three stocks (one each from Popular, Sector-diverse, and Volatile categories), streams their prices in real time via WebSockets, and allows players to buy/sell before the game ends.
* **Integrated AI Advisor**Powered by *Ollama* and orchestrated with *LangChain*, the AI agent provides live, category-aware *buy/sell/hold* recommendations based on historical price trends and in-game context.
* **End-to-End Data Pipeline**Implements all stages of the data engineering lifecycle:
  + *Generation*: Historical market datasets, sector metadata, and player-generated events.
  + *Ingestion*: Batch imports to AWS S3, API calls to OpenFIGI, and real-time writes to PostgreSQL.
  + *Storage*: Clear separation between raw (immutable S3), processed (PostgreSQL), and event data (PostgreSQL).
  + *Transformation*: Cleaning, enrichment, and feature engineering to produce playable stock categories.
  + *Serving*: FastAPI endpoints and WebSocket streaming for gameplay and leaderboards.
  + *Consumption*: Player analytics, leaderboard generation, and AI model consumption.
* **Validation and Evaluation**Manual gameplay tests, schema and data quality checks, and AI output review confirmed the pipeline’s accuracy, data integrity, and responsiveness under target loads.

These results demonstrate that the system is not only engaging and technically sound but also structured for peer reproducibility, enabling other teams to extend it with new datasets, features, or models.

To support reproducibility, the complete source code, Docker configurations, dataset references, and detailed setup instructions are provided in the project repositories. These resources enable any technically proficient team to deploy the system in a consistent environment, replicate the results, and extend the application with minimal configuration effort.

# 2. Data engineering lifecycle

The Stock Roulette: LLM Edition pipeline was designed following the *Fundamentals of Data Engineering* lifecycle, ensuring scalability, reproducibility, and responsiveness. Figure 1 illustrates the definition of Data engineering lifecycle and Figure 2 illustrates the complete architecture, from data generation to consumption.

A diagram of data engineering lifecycle

AI-generated content may be incorrect.

Figure 1 Date Engineering Lifecycle definition of the book

## 2.0 Pipeline Diagram

A diagram of a software process

AI-generated content may be incorrect.

Figure 2 Stock Roulette data engineering pipeline, showing generation, ingestion, storage, transformation, serving and consumption stages

## 2.1 Data Generation

The project integrates both external and internal data sources to create a rich and authentic gameplay environment.

* **External sources** include historical stock market data from the Kaggle Stock Market Dataset, the S&P 500 company list from Wikipedia, and sector metadata from the OpenFIGI API
* **Internal sources** consist of player actions (stock selections, trades, final score), AI advisor recommendations, and leaderboard updates generated during gameplay.

This combination of curated external data and dynamic in-game events ensures that the simulation is both realistic and replayable.

## 2.2 Data Ingestion

Two ingestion modes power the system: batch and real-time.

* **Batch ingestion** imports the Kaggle datasets into an AWS S3 bucket, scrapes and merges Wikipedia data, and appends metadata from the OpenFIGI API.
* **Real-time ingestion** occurs during gameplay, where the FastAPI backend writes each player’s actions and AI advisor outputs directly into the PostgreSQL event layer.

By separating batch and streaming processes, the system can handle both static historical datasets and fast-moving, event-driven game data without bottlenecks.

## 2.3 Data Storage

The storage layer is organized to preserve data integrity and optimize performance:

* A **raw layer** in immutable S3 retains unaltered source files.
* A **processed layer** in PostgreSQL stores cleaned and enriched datasets.
* An **event layer** in PostgreSQL captures gameplay activity and leaderboard data for analytics and replay.

This clear separation supports traceability and makes it easy to verify or reproduce results.

## 2.4 Data Transformation

Transformation begins with cleaning: removing incomplete or invalid stock entries and filtering out companies without sector mappings after enrichment. Next, feature engineering classifies stocks into three categories: Popular, Volatile, and Sector-Diverse, based on S&P 500 membership, volatility metrics, and industry grouping.

These transformations culminate in gameplay preparation: assigning each player session a random month/year and one stock from each category.

Post-game, the event data is analyzed to compute players’ performance metrics.

## 2.5 Data Serving

The serving layer powers both the game interface and the AI advisor.

* **FastAPI endpoints** deliver processed data and manage game logic.
* **WebSockets** stream live stock prices to players with minimal latency.
* The **AI advisor** consumes historical and gameplay context to provide category-aware recommendations.

Leaderboard data is updated continuously and made available to the frontend for post-game review.

## 2.6 Additional Steps

Consumption focuses on player analytics and AI model usage. The event-layer data supports leaderboard generation, game statistics, and potential future ML models for behavior prediction.

Throughout the lifecycle, several undercurrents shape the system:

* **Data management** through schema validation and integrity checks.
* **Architecture** with cloud-hosted storage/compute separation and containerization via Docker for backend, frontend, and PostgreSQL.

By following this structured yet flexible lifecycle, the project achieves low-latency performance, reproducibility, and a clear path for future scalability.

# 3. Evaluation

The Stock Roulette application was evaluated to confirm both the **functionality** of its data engineering pipeline and the **usability** of the game for its target audience. The evaluation process focused on verifying that each stage of the pipeline operated correctly, data integrity was maintained, and real-time responsiveness met gameplay requirements.

## 3.1. Pipeline Functionality Validation:

**Generation & Ingestion**

* Verified that all external datasets (Kaggle, Wikipedia, and OpenFIGI) were successfully ingested into S3.
* Verified that all processed data were successfully ingested into PostgreSQL.
* Verified that player-generated events flowed into PostgreSQL in real time via the FastAPI backend.

**Storage**

* Clear separation between raw (immutable S3), processed (PostgreSQL), and event (PostgreSQL) layers ensured data traceability and integrity.

**Transformation**

* Cleaning and enrichment steps successfully produced the three targeted stock categories (Popular, Volatile, Sector) used in gameplay, with feature engineering validated against source metadata.

**Serving**

* FastAPI endpoints reliably served processed data and streamed real-time prices to the frontend via WebSockets.
* The AI advisor consumed the same pipeline outputs to generate contextual buy/sell/hold advice.
* Validated that the AI advisor received the correct contextual inputs and returned recommendations without breaking game flow.

**Consumption**

* Leaderboard standings and player statistics were derived from event-layer data, demonstrating the pipeline’s capacity for analytics and future ML use cases.
* Verified that event-layer data could be queried post-game for analytics.

## 3.2 Data Quality Checks:

* Datasets were validated for schema consistency, completeness, and accuracy against source references. OpenFIGI enriched sector mappings were verified to ensure no unmapped stocks persisted into the game’s stock pools.
* Verified that all records contained sector data and valid historical prices after enrichment.

## 3.3 AI Advisor Output Review:

* Spot-checked AI-generated recommendations for logical consistency given recent price trends and stock category.

## 3.4 System Performance:

* Conducted test runs with multiple concurrent sessions to assess throughput and responsiveness.
* Observed stable performance for target concurrent user levels, with consistent WebSocket delivery and leaderboard updates

These results confirm that the pipeline, from data generation to consumption, is functional, reliable, and produces accurate, actionable outputs for both gameplay and analysis. The modular design also allows future teams to reproduce or extend the system with minimal configuration changes.

# 4. Limitations and possible next steps

While the Stock Roulette pipeline is functional and modular, several constraints limit its current scalability, adaptability, and analytical depth. These limitations, along with proposed next steps, are outlined below.

## 4.1 Limitations

* The historical datasets do not include macroeconomic indicators, corporate events, or news sentiment, which restricts the realism and analytical richness of the simulation.
* The pipeline lacks a systematic method for measuring the accuracy or relevance of AI advisor recommendations. Without labeled outcomes or controlled benchmarks, it is difficult to assess model performance.
* The event layer captures gameplay data for analytics, but there is no persistent player account system, limiting longitudinal analysis of player behavior.
* The ingestion layer is currently optimized for batch historical data. Real-time ingestion of live market data is not yet integrated, reducing adaptability for hybrid simulations.

## 4.2 Next Steps

Future enhancements will address these limitations, focusing on extending the lifecycle’s robustness and scalability:

* Obtain source for macroeconomics indicators, corporate events and news sentiment to add contextual data.
* Dedicated processing instancefor continuous data ingestion, for example live market data ingestion from APIs and the data mentioned above.
* Integrate Apache Kafka to capture and process real-time market data streams alongside gameplay events.
* Usage of Apache Kafka for game insights streaming.
* Capture AI advisor outputs to database for future analysis.

# 5. Conclusion

The *Stock Roulette: LLM Edition* project demonstrates how an AI-driven application can be built on a robust, modular data engineering pipeline to deliver a responsive, educational and engaging simulation of stock trading. By combining authentic historical market data with a large language model–powered advisor, the system provides both a practical training tool and a proof-of-concept for integrating AI into real-time, data driven decision making applications.

From a data engineering standpoint, this work illustrates the value of clearly defined lifecycle stages: generation, ingestion, storage, transformation, serving, and consumption, and the importance of separating raw, processed, and event layers to maintain data traceability and integrity. The use of Docker for containerizing the backend, frontend, and database components ensures reproducibility and simplifies deployment for peer teams seeking to replicate or extend the system.

The project also highlights lessons applicable beyond finance: low-latency system design, reproducible environments, modular architecture, and data quality assurance are critical for any domain requiring AI integration with streaming data. These design principles make the application adaptable to other scenarios, such as sports analytics, IoT monitoring, or educational simulations.

Ultimately, the success of Stock Roulette: LLM Edition rests not solely on its AI component, but on the solid data engineering foundation that enables the AI to operate effectively. By uniting strong engineering practices with a clear educational purpose, the project serves as both a functional game and a model for future AI-powered, data-intensive applications.

# 6. Shareable Resource

The following resources are provided to enable peers, instructors, or other interested teams to review, reproduce, or extend the Stock Roulette: LLM Edition project.

**Code Repositories**

* Data Processing: <https://github.com/Gautham-Nagaraj/Data608-Project/tree/main/data_science>
* Backend Code: <https://github.com/Gautham-Nagaraj/Data608-Project/tree/main/game-api>
* Frontend Web Application Code: <https://github.com/Gautham-Nagaraj/Data608-Project/tree/main/stock-roulette-fe>

**Data Sources**

* *Kaggle Stock Market Dataset:* <https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset>
* *S&P 500 Companies List (Wikipedia):* <https://en.wikipedia.org/wiki/List_of_S%26P_500_companies>
* *Sector Metadata (OpenFIGI API):* <https://www.openfigi.com/api>

**Demonstrations**

* *Video Demo of Gameplay:* <https://youtu.be/RYSgN1SzpMU>

**Deployment Links** *(if applicable)*

* *Live Application:* Due to the nature of AWS education account, the EC2 instances are shut down and IP addresses are deallocated, meaning that live applications are short lived and can only be accessed when user is actively using the instances.  
  Having that said, the way to access the web application is via *http://<frontend-public-host-ip>:5173* and *http://<public-host-ip>:5173/admin*
* *API Documentation:* Due to the nature of AWS education account, the EC2 instances are shut down and IP addresses are deallocated, meaning that live applications are short lived and can only be accessed when user is actively using the instances.  
  Having that said, the way to access the web application is via *http://<backend-public-host-ip>:8000*