Stock Roulette: LLM edition

A Distributed, Data-intensive Web application

**DATA 608 – Summer 2025**

Franco Doi

Gautham Nagaraj

# Problem Statement

The rapid convergence of **large language models (LLMs)**, **real-time data processing**, and **interactive web-scale systems** has fundamentally reshaped how intelligent applications are designed and deployed. LLMs such as GPT, Ollama and Mistral, when integrated into modern data architectures, have enabled new forms of context-aware assistance across domains ranging from healthcare to finance *(LangChain, n.d.; Ollama, n.d.)*. In parallel, the demand for applications that support high user concurrency and responsive behavior under load has emphasized the importance of scalable, distributed architectures *(Amazon Web Services, n.d.-a, n.d.-b)*.

This project explores the intersection of these trends by implementing **Stock Roulette: LLM Edition** - a distributed, data-intensive web application that simulates real-time stock trading and incorporates an embedded financial advisor powered by an LLM. The system emphasizes both high-throughput data handling and interactive, conversational user support.

The project fulfills three overarching objectives:

**Educational**: To demonstrate practical data engineering and deployment skills - including ingestion, preprocessing, storage in cloud infrastructure, and live simulation logic - within the context of a real-world use case.

**Experimental**: To integrate emerging technologies such as local LLMs (via **Ollama**), an agent-based reasoning layer (via **LangChain**), and a **multi-component protocol (MCP)** server for orchestrating prompt handling and AI context.

**Entertainment**: To deliver an engaging user experience by gamifying stock simulation, allowing users to interact with historical data and AI support through a time-compressed, high-interaction interface.

**Stock Roulette** operates as a data-intensive system that simulates market behavior in real-time while supporting multiple concurrent users. A key technical challenge lies in achieving acceptable **latency and throughput** when running the LLM-based agent on a small EC2 instance, especially under user load.

The general implementation approach involves provisioning one or more **Amazon EC2 instances**, **S3 buckets**, and **persistent volumes** to host the application and data pipelines. If time permits, core services will be containerized using **Docker** to improve modularity and portability. The LLM agent solution will be deployed via **Ollama**, with **LangChain** for prompt management and tool integration.

Gameplay begins with users selecting **three stocks -** one each from the predefined categories: *Popular*, *Sector-diverse*, and *Volatile*. Users also choose a specific **month and year**, which defines the historical window for the simulation. The simulation compresses one calendar month of stock activity into a **five-minute interactive session**, during which players may **buy and sell** their selected assets. A built-in **AI advisor** (LLM-powered) offers context-aware financial commentary throughout the session. Final results are scored and posted to a **leaderboard**, with the entire experience delivered through a real-time, web-based interface.

# Data Source and Quality

The primary dataset used in this project is the "Stock Market Dataset" from Kaggle, which includes historical daily stock prices for all tickers actively traded on NASDAQ. The data was collected as follows:  
- Ticker list sourced from nasdaqtrader.com  
- Historical price data retrieved via Yahoo Finance API using the yfinance Python package  
  
The dataset spans from 1999 to April 1, 2020, offering over two decades of historical trading data. It includes daily open, high, low, close, and volume metrics for thousands of securities.

The dataset can be found at <https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset>   
  
An additional metadata file, symbols\_valid\_meta.csv, includes ticker information, financial status, and categorization. This is enriched with sector data scraped from Wikipedia’s S&P 500 list. Although the raw dataset totals a few GB, compression is unnecessary, as an EC2 instance can easily process it in memory. The enriched data set used during gameplay will be significantly smaller and will be stored in a relational database for fast retrieval.

The Wikipedia’s page with S&P 500 list can be found at <https://en.wikipedia.org/wiki/List_of_S%26P_500_companies>

# Data Storage Plan

The project adopts a tiered storage strategy optimized for different phases of the workflow:

* Raw and transformed CSV files are stored in Amazon S3 buckets.
* A Python preprocessing tool reads raw files from S3, merges in sector data, filters invalid records, and writes the final dataset to MySQL.
* The MySQL database supports:
  + Fast retrieval of stock metadata during game setup
  + Persistent storage of user sessions and leaderboard results

The raw files will remain in CSV format, and compression is avoided due to the small size of processed files and the memory capacity of the EC2 instance.

# Tools, Technologies, and Infrastructure

Languages and Libraries:

* Python for all backend services, data preprocessing, and agent logic
* Pandas, yfinance, BeautifulSoup for data collection and enrichment
* LangChain + Ollama for LLM-based financial advisor chatbot
* SQLAlchemy or MySQL connector for DB access
* Vue.js for frontend

Cloud Infrastructure:

* Amazon EC2: A single instance will host the entire stack (backend, frontend, LLM, database, agent)
* Amazon S3: Object storage for raw and intermediate datasets
* MySQL: Relational database for enriched ticker metadata and game results

System Design:

* Backend API (FastAPI) for game logic, leaderboard, and admin actions
* Frontend (Vue.js) for stock selection, simulation UI, and chat interface
* Admin dashboard (Vue.js ) for managing datasets, configurations, and leaderboard moderation
* MCP Server (FastMCP) for message orchestration between the chat interface and Ollama

# Success Criteria

The project will be considered successful if the following criteria are met:

1. All components function end-to-end: from dataset ingestion and enrichment to gameplay and leaderboard recording.
2. Users can complete the 5-minute simulation without frontend or backend failure.
3. The AI agent may hallucinate but must remain contextually responsive within its knowledge cutoff.
4. Game performance is accurately calculated and reflected on the leaderboard.
5. Multiple users can play concurrently without performance degradation.
6. The admin dashboard works for data upload, configuration, and moderation.
7. Both players and admins can access the system via public HTTPS, ensuring secure communication.

# Team Roles and Responsibilities

All project responsibilities are shared collaboratively by both team members:

* Data Engineer: Franco, Gautham
* Backend Developer: Franco, Gautham
* Frontend Developer: Franco, Gautham
* AI Engineer / Agent Dev: Franco, Gautham
* DevOps Engineer: Franco, Gautham
* Project Manager: Franco, Gautham

# References

1. Jacksoncrow. (2020). *Stock Market Dataset* [Dataset]. Kaggle. <https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset>
2. NASDAQ Trader. (n.d.). *Symbol Directory Files*. <https://www.nasdaqtrader.com/trader.aspx?id=symboldirdefs>
3. Yahoo Finance. (n.d.). *Historical Stock Data*. <https://finance.yahoo.com>
4. yfinance developers. (n.d.). *yfinance Python Package*. <https://github.com/ranaroussi/yfinance>
5. Wikipedia contributors. (2024). *List of S&P 500 Companies*. Wikipedia. <https://en.wikipedia.org/wiki/List_of_S%26P_500_companies>
6. LangChain Developers. (n.d.). *LangChain Documentation*. <https://www.langchain.com/>
7. Ollama Developers. (n.d.). *Ollama: Run LLMs Locally*. <https://ollama.com/>
8. Amazon Web Services. (n.d.). *Amazon EC2 Documentation*. <https://docs.aws.amazon.com/ec2/>
9. Amazon Web Services. (n.d.). *Amazon S3 Documentation*. <https://docs.aws.amazon.com/s3/>
10. Mistral <https://mistral.ai/>