SwingSage by Team 1

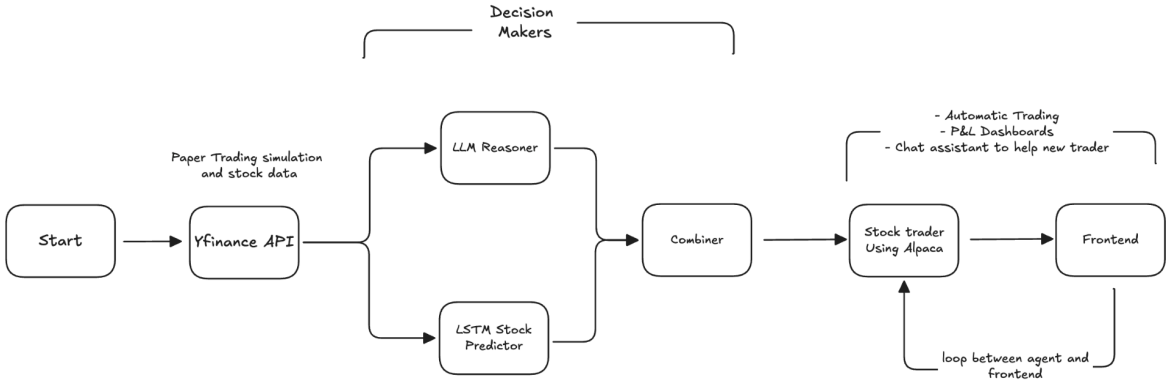
Team Members:

- Max Xie
- Joseph Alhambra
- Atharva Walawalker

Problem statement

Retail traders drown in fragmented signals: delayed fundamentals, noisy sentiment, and opaque quant alerts make it hard to act with conviction. We built SwingSage to close that gap with a single agent that digests streaming market data, and predictive analytics, then surfaces an actionable trade directive. Each recommendation ships with human-readable reasoning and historical evidence so the operator can decide quickly without reverse-engineering the models.

Architecture



The diagram explains our overall system architecture, to implement the system's core trading logic, our team designed a data pipeline that initiates with a user query for a specific stock. We utilized the Yfinance API to retrieve the necessary market data, routing it in parallel to two distinct modules: an LSTM Stock Predictor for quantitative time-series forecasting and an LLM Reasoner for qualitative market analysis. We configured both models to output a standardized [BUY, HOLD, SELL] probability vector. These vectors are aggregated through a trained combiner model to generate a final, high-confidence signal. This signal is subsequently transmitted to the Alpaca-based trading agent, which is set up to either execute autonomous paper trades or send informative insights to the frontend dashboard and chat assistant.

Working

LSTM Decision Module

We chose to train a multi-layer LSTM neural network designed to predict next-day stock prices using past closing prices from yfinance. The LSTM serves strictly as a regression model, predicting future price values while a separate decision layer interprets those predictions to generate Buy/Sell/Hold trading signals.

The model uses 2 years of AAPL stock data, with prices normalized using MinMax scaling. Each training sample consists of the previous 100 days of closing prices, allowing the model to learn temporal patterns and trends. The data was split chronologically: the first 65% for training and the latest 35% reserved for testing.

The network architecture is a stacked LSTM, having three LSTM layers (50 nodes each) followed by a Dense output layer that predicts a single price value. The model is trained with Mean Squared Error (MSE) loss and the Adam optimizer, using 100 epochs and batch size of 64.

Performance is evaluated using R^2 and MAPE on inverse-scaled predictions. Results show strong trend-tracking capability, with Test $R^2 \approx 0.88$ and MAPE $\approx 3\%$, showing that the model reliably follows the direction and shape of real price movements. Since the trading strategy depends primarily on trend accuracy rather than exact price accuracy, we thought this would be good for producing meaningful buy/sell signals.

After generating predictions, a post-processing function analyzes predicted price changes and assigns Buy, Sell, or Hold signals based on percentage thresholds.

LLM Decision System

The LLM decision module gathers a 60-day snapshot of price action, technical indicators, and any user notes before calling the DeepSeek model. The prompt asks for BUY, SELL, and HOLD scores that add up to one plus a short, plain-language explanation. We store both the probabilities and the text so the combiner and the human reviewer can see exactly why the model leans one way or another.

Dialing that in took plenty of trial and error. We started with the newest Deepseek Model, DeepSeek V3.2. It looked great in benchmarks, but since it is a reasoning model, it needed almost a minute per trade, so we switched to the quicker V3.1 release

that still performs well on finance tasks. The dataset generator showed that indicators like SMA, RSI, MACD, Bollinger Bands, and OBV beat raw tables—tabular dumps made the model wander, while the engineered signals produced steady JSON responses. Those lessons shaped the prompt that now powers every LLM call.

Model Decision Combiner

To generate more robust trading signals, we designed a combining model to merge the distinct predictive capabilities of our LLM and LSTM modules. We implemented a weighted average mathematical model formulated as **alpha * LLM + beta * LSTM** (alpha beta are both trainable). This approach allows us to mathematically balance the qualitative insights of the Large Language Model with the quantitative forecasting of the LSTM to create a unified prediction vector.

To optimize this balance, the architecture utilizes two learnable parameters trained using the **Adam optimizer** and **Negative Log-Likelihood Loss (NLLLoss)**. Through this training process, the model converged on optimal weight values of **alpha=0.21** and **beta=1.33**. These specific values indicate that the system learned to place a higher degree of trust in the LSTM's historical data processing for this configuration compared to the LLM's reasoning.

The effectiveness of this ensemble approach was validated by our performance metrics. The combiner achieved a **46.67%** accuracy rate over a **90-day** simulation period. This represents a clear improvement over the standalone Large Language Model, which yielded an accuracy of only **32.22%** in the same test environment, effectively demonstrating the value of merging the two signal sources.

For the final decision execution, the system follows a sequential processing pipeline. It first computes the weighted sum of the inputs and applies a log-softmax normalization. We then use an argmax function to isolate the highest probability action, resulting in a distinct [BUY, SELL, HOLD] command that is transmitted to the trading agent.

Combined Agentic Autotrader

The combined agent runs the full daily loop: download yfinance data, refresh the indicators, call the LSTM, call the LLM, blend their BUY/SELL/HOLD vectors with the alpha–beta weights, and push the final action through our FastAPI server. Guardrails such as DRY_RUN, position size caps, and log streaming keep every trade visible to the operator.

Earning that loop's trust took the same kind of grind. The generator script hit the LLM for dozens of historical dates, wrote out the predictions, and tagged ground truth with the simple ±0.5% price move rule. We replayed those CSVs multiple times to settle on the weights that now sit in production. Wiring the agentic server to the UI also took a lot of tries. We struggled to connect multiple modules together, as well as handle Alpaca errors, dashboard connection issues, and chat updates. It also took repeated Cloudflare tunnel tweaks plus Vercel rebuilds before the frontend, backend, and Alpaca paper broker were finally able to communicate in real time.

Drawbacks and Future Scope

Drawbacks:

- The current system only works for selected tickers we trained on like AAPL, so it lacks generalisation to the rest of the market.
- The accuracy of the system is barely better than random guessing, so larger trained models could help this.
- We are using Colab for compute which causes huge latency issues for real-time inference.

Future Scope:

- Train a larger LSTM with way more tickers in the training data.
- Build a larger combiner model with more than 90 days worth of training samples.
- The LLM can use market sentiment as an input feature.
- Maybe finally try the model on actual trading and not just paper trades once it is good enough.