# Project Report: NeuralTrade Quantitative Trading Ecosystem

**Developer:** Cyrus Amalan

**Project Type:** Full-Stack Algorithmic Trading Platform

**Core Stack:** Python, Machine Learning (Scikit-Learn), PostgreSQL, Flask, JavaScript

---

## 1. Executive Summary

NeuralTrade is a proprietary, full-stack automated trading platform designed to bridge the gap between quantitative research and live execution on the Robinhood brokerage. Unlike simple rule-based bots, NeuralTrade employs an ensemble machine learning architecture—dubbed the **"Council of Strategies"**—which aggregates technical signals from multiple distinct algorithms to feed a Random Forest meta-model. The system includes a custom-built walk-forward backtesting engine, a real-time ingestion pipeline that overcomes API limitations for futures data, and a robust analytics dashboard for post-trade performance tracking.

## 2. Quantitative Architecture: "The Council" Ensemble

The core intelligence of the system relies on a two-tier decision-making process designed to reduce overfitting and adapt to changing market regimes.

#### Tier 1: Feature Extraction (The Council)

Instead of feeding raw price data directly into a model, the system first passes market data through a "Council" of four distinct technical strategies defined in `strategies.py`. Each strategy casts a normalized vote (-1, 0, 1):

- **Mean Reversion:** Uses RSI (14) to detect overbought/oversold conditions (<30 or >70).
- **Trend Following:** Compares price action against Volume-Weighted Average Price (VWAP) to determine bullish control.
- **Volatility Breakout:** Utilizes Bollinger Bands (20, 2) to identify price expansion events.
- **Momentum:** Monitors MACD line vs. Signal line crossovers to detect momentum shifts.

#### Tier 2: The Meta-Model (Random Forest)

A **Random Forest Classifier** serves as the decision engine. It ingests the "Council Votes" alongside raw market features (Volatility, Relative Strength vs. S&P 500) to predict the probability of a positive return over the next execution window (e.g., 15 minutes or 1 day).

- **Dynamic Retraining:** The model is retrained on a rolling window (e.g., every 5 days) to adapt to recent market behavior, preventing model drift.

## 3. Engineering Highlights & Challenges

### A. Hybrid Data Ingestion Pipeline

- **The Challenge:** The Robinhood API does not provide public endpoints for "Robinhood Legend" (Futures) data, creating a blind spot for multi-asset tracking.
- **The Solution:** Engineered a custom hybrid synchronization engine in `sync_trades.py`. It combines standard API polling for Stocks/Crypto with a bespoke CSV parsing module that ingests manually exported Futures data. This allows for a unified database view across all asset classes despite API fragmentation.

### B. Custom Simulation Engines

Built two distinct simulators to validate strategies before deployment:

- **Intraday Simulator:** A high-frequency engine (`day_engine.py`) that processes 5-minute candles. It includes realistic constraints like "Force Close" logic at 3:55 PM EST to eliminate overnight risk.
- **Hindsight Analytics:** The engine calculates "Opportunity Cost" by flagging historical moments where the stock rose >1% but the AI confidence score was too low to trigger a buy, enabling continuous feedback for model tuning.
- **Walk-Forward Backtester:** Implemented a rolling-window training method in `bot_engine.py` to simulate real-world trading conditions, strictly preventing data leakage by ensuring the model only trains on past data while testing on future data.

### C. Persistence & State Management

- **Database Schema:** Designed a normalized **PostgreSQL** schema (`trade_journal`) that tracks the full lifecycle of a trade. It pairs "Buy" and "Sell" orders using FIFO (First-In-First-Out) logic to accurately calculate realized P&L and adjust for regulatory fees (SEC/TAF).
- **Orphan Handling:** Implemented robust logic to handle "Orphan Sells"—instances where a sell order executes without a linked buy in the database—by queuing them for manual reconciliation or auto-matching against open positions.

## 4. Interactive Analytics Dashboard

A responsive web interface built with **Flask** and **Tailwind CSS** provides real-time observability into system performance.

- **Market Replay Mode:** A JavaScript-based tool that redraws historical trading days candle-by-candle. This allows for visual "debugging" of the algorithm's decisions against price action in real-time.
- **P&L Heatmap:** A visual calendar aggregating daily trade volume and net profit, allowing for rapid identification of profitable streaks and drawdowns.

## 5. Technology Stack

| Domain | Technologies |
| --- | --- |
| **Languages** | Python (3.13+), JavaScript (ES6+), SQL |
| **Machine Learning** | Scikit-Learn (Random Forest), Pandas-TA, Numpy |
| **Backend & API** | Flask, Robinhood API (`robin_stocks`), yFinance |
| **Data Storage** | PostgreSQL (Relational Data), Joblib (Model Serialization) |
| **Frontend** | HTML5, Tailwind CSS, Plotly.js (Visualization) |
| **DevOps/Tools** | Git, Conda, CSV Data Pipelines |

## 6. Conclusion

NeuralTrade demonstrates the ability to architect complex, full-stack financial systems. By combining rigorous quantitative research methods with robust software engineering practices, the platform solves key challenges in data ingestion, model validation, and automated execution, resulting in a disciplined and scalable trading operation.