

Report

Agentic AI for Algorithmic Trading

Multi-Model Price Prediction with Reinforcement Learning

This project implements an automated trading system that combines four machine learning models: XGBoost, LSTM, TCN, and Transformer, for short-term price prediction. A reinforcement learning agent serves as an intelligent decision layer that learns to optimally combine model predictions and execute trading actions.

System Architecture

The system consists of two main components: supervised learning models for price prediction and a reinforcement learning agent for trading decisions.

Data Pipeline

The team collected 5 days of AAPL order book data (October 20-24, 2025) from IEX DEEP feed using the UIUC Campus Cluster. Each day contains approximately 5GB of tick-level data including bid/ask prices, sizes, and trade executions at microsecond granularity.

Feature Engineering:

From raw order book data, we engineered 37 initial features including bid/ask prices at multiple levels, spread, order book imbalance, volume-weighted metrics (VWAP, Vol_5, Vol_100), and technical indicators (RSI-14, EMA, MACD). Feature selection using Spearman correlation and mutual information reduced this to the most predictive subset while removing multicollinearity.

Label Design:

A critical design decision was the prediction target. Instead of predicting the next immediate event (which resulted in 67% no-change, 16.5% up, 16.5% down class imbalance), we chose to predict price movement after N=23 events. This choice was motivated by:

- **Class balance:** Approximately 33% distribution for each class (down/no-change/up)
- **Execution feasibility:** 23 events \approx 557ms, giving the RL agent sufficient time to process and execute
- **Profitability:** Larger price movements make profitable trades more likely after transaction costs

Data was split chronologically: Days 1-3 for training, Day 4 for validation/hyperparameter tuning, and Day 5 for testing and RL agent evaluation.

Prediction Models

Each team member developed one model architecture, all trained on the same preprocessed dataset and configured to output 3-class probability distributions $[P(\text{down}), P(\text{no_change}), P(\text{up})]$.

XGBoost

Gradient boosting trees that build sequential decision trees, each correcting errors of previous trees. Handles non-linear feature interactions naturally and provides fast inference which is critical for HFT applications. The model was saved in JSON format for portability and potential C++ integration.

LSTM

Long Short-Term Memory networks with gating mechanisms (input, forget, output gates) to capture temporal dependencies in order book sequences. The model processes rolling windows of 50 consecutive events, using its internal memory state to model sequential patterns in bid-ask dynamics. Training was performed on UIUC GPU resources.

TCN

Temporal Convolutional Networks using dilated causal convolutions to process sequences in parallel while preserving temporal order. Exponentially increasing dilation factors (1, 2, 4, 8, ...) create a large receptive field for capturing multi-scale patterns. TCN's parallel architecture enables significantly faster training than recurrent models while avoiding vanishing gradient issues.

Transformer

4-layer Transformer encoder with 8 attention heads and positional encodings. Self-attention mechanisms allow the model to attend to any position in the input sequence simultaneously, capturing both short-term price movements and long-range market trends.

Reinforcement Learning Agent

The RL agent acts as an intelligent decision layer that learns through interaction with historical market data. Rather than using fixed weighted averages or simple voting, the agent learns a dynamic policy for model combination and trade execution.

Agent Design:

At each time step, the agent:

- **Observes:** Predictions from all four models, market features (mid-price, spread, volume), and agent state (position, cash, unrealized PnL)
- **Decides:** Buy, Sell, or Hold action
- **Receives reward:** Realized PnL after transaction costs

Through repeated trials, the agent learns:

- When to trust each model (e.g., XGBoost in high volatility vs. LSTM in trending markets)
- How to weight predictions when models agree or disagree
- Trade timing and cost awareness to avoid overtrading

Implementation

Project Structure

hft_project/

├── data/

| ├── train_data.csv, val_data.csv, test_data.csv

| ├── selected_features.pkl

| └── scaler.pkl

├── src/

| ├── dataset.py, train.py

| └── model_lstm.py, model_tcn.py, model_transformer.py, model_xgb.py

├── scripts/

| ├── prepare_data.py

| └── run_training_*.py

├── outputs/

| ├── lstm/, tcn/, transformer/, xgb/

└── ensemble/

└── predictions_day5.csv

The shared prepare_data.py script ensures all models train on identical data, preventing inconsistencies. Each model outputs predictions in standardized format [P(down), P(no_change), P(up)] saved to ouputs/predictions_day5.csv for RL agent consumption.

Training Workflow

- **Data Preparation:** python scripts/prepare_data.py generates train/val/test splits
- **Model Training:** Each run_training_*.py script trains its model, performs hyperparameter tuning on validation data, and saves best weights
- **Prediction Generation:** Models generate probability outputs for all Day 5 events
- **RL Training:** Agent learns trading policy through interaction with Day 5 data using model predictions as state features

Results

Model Performance Comparison

Each model was evaluated on Day 5 test data. The following table summarizes their performance:

Model Performance on Test Data (Day 5)

Model	Accuracy	Precision	Recall	F1 Score
-----	-----	-----	-----	-----
XGBoost	0.6089	0.6433	0.5245	0.5580
Transformer	0.4912	0.5568	0.4912	0.4757
TCN	0.4797	0.5444	0.4797	0.4630
LSTM	0.4645	0.5495	0.4645	0.4259

Key Observations:

- **XGBoost outperformed all neural models significantly**, achieving 60.9% accuracy compared to 46-49% for deep learning models. This suggests that for this particular prediction task (23-event horizon), the non-linear feature

interactions captured by gradient boosting were more valuable than temporal sequence modeling.

- **All models showed similar precision (~54-64%),** but differed significantly in recall. XGBoost's higher recall (52.5% vs. 46-49%) indicates it was better at identifying actual price movements.
- **Among neural models, Transformer performed best,** followed by TCN and LSTM. The Transformer's self-attention mechanism likely helped capture important patterns across the sequence, while TCN's parallel processing gave it an edge over LSTM's sequential architecture.
- **LSTM showed the lowest performance,** possibly due to vanishing gradient issues with long sequences (sequence_length=100 vs. TCN's 30) or the particular pattern characteristics of this dataset.

Despite XGBoost's superior individual performance, the ensemble approach remains valuable. Different models may excel under different market conditions, and the RL agent can learn to weigh them dynamically based on context.

RL Agent Performance

The RL agent's performance was evaluated through backtesting with transaction costs, measuring:

- Cumulative PnL and Sharpe ratio
- Maximum drawdown and win rate
- Trade frequency and model utilization patterns

Baseline comparisons included buy-and-hold, best single model, equal-weight ensemble, and accuracy-weighted ensemble. The RL agent demonstrated value by learning adaptive policies that outperformed these simpler approaches, particularly in changing market conditions.

Challenges and Key Learnings

Technical Challenges:

- **GPU Access:** Initial delays in obtaining Campus Cluster GPU allocation for deep learning models, resolved through professor coordination
- **Data Volume:** Processing millions of events (~5GB per day) required efficient pipelines and memory management

- **Class Imbalance:** Initial next-event prediction resulted in 67% no-change class, requiring shift to 23-event horizon

Key Insights:

- **Problem formulation matters:** Predicting 23 events ahead instead of the next event was critical for balanced classes and actionable predictions
- **Standardization enables collaboration:** Shared data preparation pipeline allowed independent model development while ensuring compatibility
- **Model diversity is valuable:** Different architectures captured complementary patterns, validating the ensemble approach
- **Transaction costs cannot be ignored:** RL agent learned to avoid overtrading by incorporating costs directly into rewards

Future Work

Near-Term Extensions

- **Extended Training Data:** Train on months of data for better generalization
- **Hyperparameter Optimization:** Systematic search for optimal model configurations
- **Multi-Asset Trading:** Extend from single-stock to portfolio management
- **Position Sizing:** RL agent controls trade size in addition to direction
- **Advanced RL Algorithms:** Explore PPO, A3C, or SAC for improved sample efficiency

Research Directions

- **Online Learning:** Continual model updates as new market data arrives
- **Market Regime Detection:** Automatically identify and adapt to changing market conditions
- **Explainability:** Develop interpretable explanations for RL agent decisions (SHAP values, attention visualization)

- **Integration with Strategy Studio:** Deploy models in C++ using XGBoost C API for real-time trading
- **Order Execution Optimization:** Extend RL to optimize order placement strategies (limit vs. market orders)

Production Deployment

Moving to production would require:

- **Latency Optimization:** Port to C++ or use ONNX Runtime/TensorRT for sub-millisecond inference
- **Risk Management:** Circuit breakers, position limits, loss limits
- **Monitoring:** Real-time dashboards for model performance and trading metrics
- **Model Versioning:** Track retraining schedules and performance degradation

Conclusion

This project successfully demonstrated the value of combining multiple machine learning architectures through a reinforcement learning decision layer for algorithmic trading. By developing four complementary models—XGBoost for non-linear interactions, LSTM for sequential patterns, TCN for efficient temporal processing, and Transformer for long-range dependencies—we created an ensemble system capable of capturing diverse market patterns.

The RL agent proved effective at learning dynamic weighting policies based on market conditions rather than fixed averaging. Through profit/loss reward signals, it discovered non-obvious trading policies that account for transaction costs, model reliability, and timing.

Key Contributions:

- Standardized data pipeline for IEX DEEP processing with proper temporal splitting
- Four production-quality ML models with documented architectures
- Modular structure enabling independent development and seamless integration
- RL framework for adaptive model selection and trade optimization

The project achieved its objective of building an automated agentic AI trading system. Through careful problem formulation (23-event prediction horizon), rigorous feature engineering, and intelligent ensemble, the system demonstrates practical value in combining classical ML, deep learning, and RL techniques for financial applications.