Advanced Modular Cryptocurrency Trading & Analytics Platform: A Comprehensive Mathematical and Implementation Framework

Sineshaw (Shaw) Mesfin Tesfaye

Department of Computer Science
Georgia State University
stesfaye4@student.gsu.edu

2026

Cryptocurrency Trading Research — Integrated Submission

Methods Transparency Checklist

Category	Details
Data & Instruments	Coinbase Advanced Trade API; BTC-USD, ETH-USD, SOL-USD, and 370+ altcoin pairs; 1-min to daily granularity; Jan 2023–Feb 2026
Targets & Horizons	1–4 hour price direction classification; next-candle price regression; multi-horizon forecasting (5-min, 1-hr, 4-hr)
Cross-Validation	Walk-forward validation with expanding window; 70/15/15 train/validation/test split; purged k-fold to prevent look-ahead bias
Costs & Slippage	0.6% round-trip transaction cost (Coinbase Advanced); 10 bps slippage model; market impact via square-root model
Risk Controls	Maximum 25% portfolio risk per position; Kelly criterion + volatility-adjusted position sizing; GARCH/VaR-based TP/SL with regime detection; 10-step advanced TP/SL pipeline

Abstract

This paper presents a comprehensive, modular cryptocurrency trading and analytics platform that integrates advanced machine learning, quantitative finance, and real-time market data processing. The system employs a stacking ensemble of Random Forest, XGBoost, Support Vector Regression, Ridge, and Lasso regressors with a gradient boosting meta-learner, achieving directional accuracy exceeding 62% on out-of-sample cryptocurrency data. We develop 48 WorldQuant-style alpha factors spanning price-based, momentum, cross-sectional, statistical, and high-frequency categories. Risk management is formalized through GARCH(1,1) volatility modeling, Value at Risk with Expected Shortfall, Kelly criterion position sizing with volatility-adjusted blending, and factor model risk decomposition. The live execution pipeline integrates the stacking ML engine for signal generation, the risk manager for Kelly/volatility-based position sizing, and a 10-step advanced TP/SL engine using GARCH forecasting, regime detection, and liquidity analysis. Signal denoising via Kalman filtering and wavelet transforms improves feature quality. Walk-forward validation and Kupiec backtesting confirm model robustness across bull, bear, and sideways market regimes. The platform processes live WebSocket feeds from Coinbase Advanced Trade API and executes trades through an event-driven architecture with sub-second latency. Three new subsystems—advanced technical analysis with support/resistance and Fibonacci levels, a futures trading engine with dynamic leverage, and a sentiment analyzer—extend the core platform with independently toggleable capabilities. All mathematical foundations, implementation details, and empirical results are documented for full reproducibility.

Keywords: cryptocurrency trading, machine learning, ensemble methods, alpha factors, risk management, GARCH, Kelly criterion, stacking ensemble, quantitative finance, Coinbase API, support/resistance analysis, sentiment analysis, futures leverage

Reproducibility Statement

All source code, configuration files, and trained model artifacts are available in the project repository. Data is sourced exclusively from public Coinbase API endpoints. Random seeds are fixed for all stochastic components. Database schemas and SQL queries are provided in the appendices.

Ethics Statement

This research involves automated trading of publicly listed cryptocurrency assets. No insider information or non-public data sources are used. The system includes circuit breakers, maximum position limits, and risk controls to prevent excessive market impact. All trading is conducted on the author's personal accounts.

Limitations

Results are based on historical and live trading data from a single exchange (Coinbase). Cryptocurrency markets exhibit regime changes, structural breaks, and liquidity variations that may cause future performance to differ materially from reported results. Transaction costs and slippage are modeled but may underestimate real-world execution friction during high-volatility periods. The sentiment analyzer's simulated sentiment mode (used during backtesting when historical news data is unavailable) is a heuristic approximation and may not capture all sentiment-driven market dynamics.

Contents

1 Introduction

1.1 Motivation

Cryptocurrency markets present unique challenges and opportunities for algorithmic trading systems. Unlike traditional equity markets, crypto assets trade 24/7 across fragmented exchanges with varying liquidity, exhibit extreme volatility, and are influenced by sentiment-driven dynamics that differ fundamentally from established financial instruments (**??**). These characteristics demand trading systems that combine rigorous quantitative methods with robust engineering.

1.2 Key Contributions

This work makes the following contributions:

(1) A **modular system architecture** with cleanly separated components for data ingestion, feature engineering, model training, risk management, order execution, and analytics.

(2) **48 WorldQuant-style alpha factors** organized into price-based, momentum, cross-sectional, statistical, and high-frequency categories, with formal mathematical definitions.

(3) A **stacking ensemble** combining Random Forest, XGBoost, SVR, Ridge, and Lasso base learners with a gradient boosting meta-learner.

(4) **Comprehensive risk management** integrating GARCH(1,1) volatility forecasting, parametric and historical VaR, Expected Shortfall, Kelly criterion position sizing with volatility-adjusted blending, and factor model risk decomposition.

(5) **Signal denoising** via Kalman filtering and wavelet transforms for improved feature quality.

(6) **Rigorous validation** through walk-forward testing, Kupiec VaR backtesting, bootstrap confidence intervals, and regime-conditional performance analysis.

(7) **End-to-end live pipeline integration** where the stacking ML engine drives signal generation, the risk manager computes Kelly/volatility-based position sizes, and a 10-step advanced TP/SL engine (GARCH + regime detection + liquidity analysis) calibrates exit levels—all wired into the live execution path.

(8) **Three new trading subsystems**: advanced technical analysis with support/resistance and Fibonacci levels, a futures trading engine with dynamic leverage computation, and a sentiment analyzer combining NLP and market-derived signals.

1.3 System Overview

The platform is implemented in Python and consists of the following primary modules:

- `maybe.py` — Core trading logic, indicator calculation, model training, and Dash UI.
- `flask_trading_dashboard.py` — Web-based dashboard with real-time analytics, GARCH-based TP/SL, and backtesting.
- `trading_engine.py` — Live trading loop with integrated ML signals, risk-adjusted position sizing, and advanced TP/SL execution.
- `stacking_ml_engine.py` — Stacking ensemble decision engine (RF + XGBoost + SVR $\rightarrow$ Ridge meta-learner).
- `market_scanner.py` — Market scanning across 370+ symbols with stacking ML signal generation.
- `risk_manager.py` — Kelly criterion + volatility-adjusted position sizing with 1% risk cap per trade.
- `advanced_tp_sl_engine.py` — 10-step TP/SL pipeline: GARCH volatility, regime detection, liquidity risk, VaR, forecast uncertainty.
- `integrate_advanced_tp_sl.py` — Bridge layer connecting the advanced TP/SL engine to the live execution path.
- `stefan_jansen_improvements.py` — Enhanced feature engineering with 48 alpha factors, momentum features, and denoising.
- `quantitative_finance_ml_enhancement.py` — GARCH, regime detection, Kelly criterion, and Monte Carlo simulation.
- `portfolio_var.py` — Portfolio VaR, Component VaR, and Expected Shortfall calculations.
- `factor_risk.py` — Factor model regression and risk decomposition.
- `enhanced_features.py` — Technical indicator library (RSI, Bollinger Bands, volume ratios).
- `advanced_technical_analysis.py` — Support/resistance identification, Fibonacci retracement/extension levels.
- `futures_trading_engine.py` — Leveraged futures trading with dynamic leverage computation.

- `crypto_sentiment_analyzer.py` — NLP sentiment analysis combined with market-derived condition indicators.

2 Background and Literature Review

2.1 Financial Machine Learning

The application of machine learning to financial markets has grown substantially since the seminal work of **?** on the Adaptive Markets Hypothesis. **?** provides a comprehensive treatment of ML techniques for algorithmic trading, including factor research, ensemble methods, and alternative data integration. **?** introduces innovations such as the triple barrier method, fractional differentiation, and purged cross-validation that address unique challenges in financial ML.

2.2 Ensemble Methods in Finance

Ensemble learning combines multiple base models to improve prediction accuracy and robustness. **?** introduced Random Forests as bagged decision tree ensembles. **?** developed XGBoost, a scalable gradient boosting framework that has become dominant in structured data prediction. **?** formalized stacked generalization, where a meta-learner combines base model predictions. In the cryptocurrency context, **?** demonstrates that stacking ensembles outperform individual models for trading signal generation.

2.3 Cryptocurrency Market Research

Cryptocurrency markets are characterized by high volatility, 24/7 trading, fragmented liquidity, and susceptibility to sentiment-driven price movements (**??**). **?** document that standard risk factors fail to explain cryptocurrency returns, motivating the development of crypto-specific alpha factors. **?** find that momentum and reversal effects exist in crypto markets but differ in magnitude and persistence from equity markets.

2.4 Technical Analysis and Market Microstructure

Technical analysis remains widely used in cryptocurrency trading despite mixed evidence of its efficacy in traditional markets (**?**). The key indicators implemented in our system include exponential moving averages, relative strength index, MACD, Bollinger Bands, and stochastic oscillators. Market microstructure theory (**?**) informs our modeling of price impact, bid-ask spreads, and liquidity.

2.5 Deep Learning for Time Series

Long Short-Term Memory (LSTM) networks (?) have shown promise for financial time series prediction due to their ability to capture long-range dependencies. ? demonstrate that LSTMs outperform traditional models for stock return prediction over certain horizons.

2.6 Risk Management

Modern portfolio risk management builds on the work of ? and has been extended through Value at Risk (?), Expected Shortfall (?), and the Kelly criterion (??). GARCH models (??) remain the standard for volatility forecasting in financial applications.

3 Mathematical Framework

3.1 Technical Indicators

3.1.1 Exponential Moving Average (EMA)

The exponential moving average assigns exponentially decreasing weights to past observations. For a price series $\{P_t\}$ with span parameter n:

$$\text{EMA}_t = \alpha \cdot P_t + (1-\alpha) \cdot \text{EMA}_{t-1}, \quad \alpha = \frac{2}{n+1}, \tag{1}$$

where α is the smoothing factor and $\text{EMA}_0 = P_0$. The system computes EMA_{12} and EMA_{26} as implemented in `maybe.py`:

```
df['EMA12'] = df['close'].ewm(span=12, adjust=False).mean()
df['EMA26'] = df['close'].ewm(span=26, adjust=False).mean()
```

Listing 1: EMA calculation in `maybe.py`

3.1.2 Relative Strength Index (RSI)

The RSI measures the magnitude of recent price changes to evaluate overbought or oversold conditions:

$$\text{RSI}_t = 100 - \frac{100}{1+\text{RS}_t}, \quad \text{RS}_t = \frac{\overline{\text{Gain}_t}}{\overline{\text{Loss}_t}}, \tag{2}$$

where

$$\overline{\text{Gain}_t} = \frac{1}{n}\sum_{i=0}^{n-1} \max(\Delta P_{t-i}, 0), \tag{3}$$

$$\overline{\text{Loss}_t} = \frac{1}{n}\sum_{i=0}^{n-1} \max(-\Delta P_{t-i}, 0), \tag{4}$$

with $\Delta P_t = P_t - P_{t-1}$ and $n = 14$ by default.

3.1.3 Moving Average Convergence Divergence (MACD)

$$\text{MACD}_t = \text{EMA}_{12,t} - \text{EMA}_{26,t}, \tag{5}$$

$$\text{Signal}_t = \text{EMA}_9(\text{MACD}_t), \tag{6}$$

$$\text{Histogram}_t = \text{MACD}_t - \text{Signal}_t. \tag{7}$$

3.1.4 Bollinger Bands

Bollinger Bands construct an envelope around a moving average using standard deviation:

$$\text{Middle}_t = \text{SMA}_{20,t} = \frac{1}{20}\sum_{i=0}^{19} P_{t-i}, \tag{8}$$

$$\text{Upper}_t = \text{SMA}_{20,t} + k \cdot \sigma_{20,t}, \tag{9}$$

$$\text{Lower}_t = \text{SMA}_{20,t} - k \cdot \sigma_{20,t}, \tag{10}$$

where $\sigma_{20,t}$ is the rolling 20-period standard deviation and $k = 2$ by default. The Bollinger Band position indicator normalizes the current price within the band:

$$\text{BB\_Position}_t = \frac{P_t - \text{Lower}_t}{\text{Upper}_t - \text{Lower}_t}. \tag{11}$$

3.1.5 Stochastic Oscillator

$$\%K_t = 100 \cdot \frac{P_t - L_{14,t}}{H_{14,t} - L_{14,t}}, \tag{12}$$

$$\%D_t = \text{SMA}_3(\%K_t), \tag{13}$$

where $H_{14,t} = \max_{i\in[t-13,t]} H_i$ and $L_{14,t} = \min_{i\in[t-13,t]} L_i$.

3.1.6 Average True Range (ATR)

$$\text{ATR}_t = \frac{1}{14}\sum_{i=0}^{13} \text{TR}_{t-i}, \quad \text{TR}_t = \max(H_t - L_t,\ |H_t - C_{t-1}|,\ |L_t - C_{t-1}|). \tag{14}$$

3.1.7 On-Balance Volume (OBV)

$$\text{OBV}_t = \text{OBV}_{t-1} + \text{sign}(\Delta P_t) \cdot V_t. \tag{15}$$

3.2 Advanced Statistical Features

3.2.1 Rolling Volatility

Annualized rolling volatility over window w:

$$\sigma_{w,t} = \sqrt{\frac{252}{w} \sum_{i=0}^{w-1} (r_{t-i} - \bar{r}_{w,t})^2}, \quad r_t = \ln \frac{P_t}{P_{t-1}}. \tag{16}$$

3.2.2 Maximum Drawdown

$$\text{MDD}_t = \max_{s \in [0,t]} \frac{M_s - P_s}{M_s}, \quad M_s = \max_{u \in [0,s]} P_u. \tag{17}$$

3.2.3 Sharpe Ratio

$$\text{SR} = \frac{\bar{r} - r_f}{\sigma_r} \cdot \sqrt{252}, \tag{18}$$

where r_f is the risk-free rate and σ_r is the standard deviation of returns.

3.2.4 Sortino Ratio

$$\text{Sortino} = \frac{\bar{r} - r_f}{\sigma_d} \cdot \sqrt{252}, \quad \sigma_d = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \min(r_i - r_f, 0)^2}. \tag{19}$$

3.2.5 Calmar Ratio

$$\text{Calmar} = \frac{\text{CAGR}}{\text{MDD}}. \tag{20}$$

3.3 WorldQuant-Style Alpha Factors

We implement 48 alpha factors organized into five categories: price-based, momentum, cross-sectional, statistical, and high-frequency. Each factor is standardized via cross-sectional z-scoring or rank normalization.

3.3.1 Price-Based Factors (Factors 1–10)

$$\alpha_1 = \text{rank}\left(\arg\max_{i\in[1,\text{ts}]} \text{SignedPower}(\text{returns}, 2)\right) - 0.5, \tag{21}$$

$$\alpha_2 = -1 \times \text{correlation}(\text{rank}(\delta(\log(V_t), 2)),\ \text{rank}\left(\frac{C_t - O_t}{H_t - L_t}\right),\ 6), \tag{22}$$

$$\alpha_3 = -1 \times \text{correlation}(\text{rank}(O_t),\ \text{rank}(V_t),\ 10), \tag{23}$$

$$\alpha_4 = -1 \times \text{Ts\_Rank}(\text{rank}(C_t),\ 9), \tag{24}$$

$$\alpha_5 = \text{rank}(O_t - \text{Ts\_Min}(O_t, 12)) \times \left[\text{rank}(\text{correlation}(V_t, \text{SMA}_5(V_t), 26))\right]^5, \tag{25}$$

$$\alpha_6 = -1 \times \text{correlation}(O_t,\ V_t,\ 10), \tag{26}$$

$$\alpha_7 = \begin{cases} -1 \times \text{Ts\_Rank}(|r_t|, 5) & \text{if } \text{ADV}_{20} < V_t, \\ -1 & \text{otherwise}, \end{cases} \tag{27}$$

$$\alpha_8 = -1 \times \text{rank}\left(\delta\left(\frac{(C_t - L_t) - (H_t - C_t)}{H_t - L_t} \times V_t,\ 1\right)\right), \tag{28}$$

$$\alpha_9 = \begin{cases} \delta(C_t, 1) \times (C_t - \text{Ts\_Min}(C_t, 5)) & \text{if } \min(\delta(C_t, 1), 0) > 0, \\ \delta(C_t, 1) & \text{otherwise}, \end{cases} \tag{29}$$

$$\alpha_{10} = \text{rank}\left(\begin{cases} \delta(C_t, 1) & \text{if } \min(\delta(C_t, 1), 0) > 0, \\ \delta(C_t, 1) & \text{otherwise} \end{cases}\right), \tag{30}$$

where $\delta(X_t, d) = X_t - X_{t-d}$, $\text{Ts\_Rank}(X, d)$ is the time-series rank of X over the past d periods, $\text{Ts\_Min}(X, d) = \min_{i\in[t-d+1,t]} X_i$, and ADV_{20} is the 20-period average daily volume.

3.3.2 Momentum Factors (Factors 11–20)

$$\alpha_{11} = \frac{C_t}{C_{t-5}} - 1, \tag{31}$$

$$\alpha_{12} = \frac{C_t}{C_{t-20}} - 1, \tag{32}$$

$$\alpha_{13} = \frac{C_t}{C_{t-60}} - 1, \tag{33}$$

$$\alpha_{14} = \frac{C_t}{C_{t-120}} - 1, \tag{34}$$

$$\alpha_{15} = \frac{C_t}{C_{t-252}} - 1, \tag{35}$$

$$\alpha_{16} = \frac{\partial P}{\partial t} \approx \Delta P_t = P_t - P_{t-1} \quad \text{(price velocity)}, \tag{36}$$

$$\alpha_{17} = \frac{\partial^2 P}{\partial t^2} \approx \Delta^2 P_t = \Delta P_t - \Delta P_{t-1} \quad \text{(price acceleration)}, \tag{37}$$

$$\alpha_{18} = \frac{\partial^3 P}{\partial t^3} \approx \Delta^3 P_t \quad \text{(price jerk)}, \tag{38}$$

$$\alpha_{19} = \text{VPT}_t = \sum_{i=1}^{t} V_i \cdot r_i \quad \text{(Volume-Price Trend)}, \tag{39}$$

$$\alpha_{20} = \frac{\text{VPT}_t - \text{VPT}_{t-10}}{\text{VPT}_{t-10}} \quad \text{(VPT Momentum)}. \tag{40}$$

3.3.3 Cross-Sectional Factors (Factors 21–28)

$$\alpha_{21} = \text{AD}_t = \sum_{i=1}^{t} \frac{(C_i - L_i) - (H_i - C_i)}{H_i - L_i} \cdot V_i \quad \text{(A/D Line)}, \tag{41}$$

$$\alpha_{22} = \frac{\text{AD}_t - \text{AD}_{t-10}}{\text{AD}_{t-10}} \quad \text{(A/D Momentum)}, \tag{42}$$

$$\alpha_{23} = \text{OBV}_t = \sum_{i=1}^{t} \text{sign}(r_i) \cdot V_i, \tag{43}$$

$$\alpha_{24} = \frac{\text{OBV}_t - \text{OBV}_{t-10}}{\text{OBV}_{t-10}} \quad \text{(OBV Momentum)}, \tag{44}$$

$$\alpha_{25} = \mathbb{1}\left[C_t > \max_{i \in [t-d, t-1]} H_i\right] \quad \text{(Breakout High, } d \in \{10, 20, 50\}\text{)}, \tag{45}$$

$$\alpha_{26} = \mathbb{1}\left[C_t < \min_{i \in [t-d, t-1]} L_i\right] \quad \text{(Breakout Low, } d \in \{10, 20, 50\}\text{)}, \tag{46}$$

$$\alpha_{27} = \frac{|\text{SMA}_{10,t} - \text{SMA}_{20,t}|}{C_t} \quad \text{(Trend Strength Short)}, \tag{47}$$

$$\alpha_{28} = \frac{|\text{SMA}_{20,t} - \text{SMA}_{50,t}|}{C_t} \quad \text{(Trend Strength Long)}. \tag{48}$$

3.3.4 Statistical Factors (Factors 29–35)

$$\alpha_{29} = \frac{C_t}{\mathrm{SMA}_{d,t}} - 1 \quad \text{(Relative Strength, } d \in \{10, 20, 50\}\text{)}, \tag{49}$$

$$\alpha_{30} = \frac{\bar{r}_{d,t}}{\sigma_{d,t}} \quad \text{(Sharpe-like Momentum, } d \in \{10, 20\}\text{)}, \tag{50}$$

$$\alpha_{31} = \frac{V_t}{\mathrm{SMA}_{20}(V_t)} \quad \text{(Volume Ratio)}, \tag{51}$$

$$\alpha_{32} = \mathrm{rank}(\sigma_{10,t}) \quad \text{(Volatility Rank)}, \tag{52}$$

$$\alpha_{33} = \mathrm{rank}(\mathrm{skew}_{20,t}) \quad \text{(Skewness Rank)}, \tag{53}$$

$$\alpha_{34} = \mathrm{rank}(\mathrm{kurt}_{20,t}) \quad \text{(Kurtosis Rank)}, \tag{54}$$

$$\alpha_{35} = H(r_{t-w:t}) \quad \text{(Hurst Exponent)}, \tag{55}$$

where the Hurst exponent is estimated via the rescaled range statistic:

$$H = \frac{\log(R/S)}{\log(n)}, \quad \frac{R}{S} = \frac{\max_{1 \le k \le n} W_k - \min_{1 \le k \le n} W_k}{\sqrt{\frac{1}{n} \sum_{i=1}^{n} (r_i - \bar{r})^2}}, \tag{56}$$

with $W_k = \sum_{i=1}^{k} (r_i - \bar{r})$ and $H > 0.5$ indicating trending behavior, $H < 0.5$ indicating mean reversion.

3.3.5 High-Frequency Factors (Factors 36–39+)

Additional factors designed for intraday and sub-hourly trading capture microstructure dynamics:

$$\alpha_{36} = \frac{V_t - \mathrm{SMA}_5(V_t)}{\mathrm{SMA}_5(V_t)} \quad \text{(Volume Surge)}, \tag{57}$$

$$\alpha_{37} = \frac{H_t - L_t}{\mathrm{SMA}_{10}(H - L)} \quad \text{(Range Expansion)}, \tag{58}$$

$$\alpha_{38} = \frac{C_t - O_t}{H_t - L_t} \quad \text{(Candle Body Ratio)}, \tag{59}$$

$$\alpha_{39} = \mathrm{corr}(|r_t|, V_t, 20) \quad \text{(Volume-Volatility Correlation)}. \tag{60}$$

These factors, together with additional intraday microstructure features (bid-ask spread proxies, trade intensity ratios, and order flow imbalance indicators), bring the total alpha factor count to 48. All factors are computed in `stefan_jansen_improvements.py` and integrated into the ML pipeline via `enhanced_features.py`.

3.4 Machine Learning Models

3.4.1 Random Forest

The Random Forest classifier (**?**) constructs an ensemble of B decision trees, each trained on a bootstrap sample with random feature subsets of size $\lfloor \sqrt{p} \rfloor$ for classification.

Gini Impurity. For a node with K classes:

$$G(t) = 1 - \sum_{k=1}^{K} p_k^2, \tag{61}$$

where p_k is the proportion of class k samples at node t.

Information Gain. The impurity reduction from splitting node t into children t_L and t_R:

$$\Delta G = G(t) - \frac{|t_L|}{|t|} G(t_L) - \frac{|t_R|}{|t|} G(t_R). \tag{62}$$

Ensemble Prediction. For classification:

$$\hat{y} = \text{mode}\{h_b(\mathbf{x})\}_{b=1}^{B}. \tag{63}$$

For regression:

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} h_b(\mathbf{x}). \tag{64}$$

Our implementation uses $B = 100$ trees with maximum depth 10:

```
RandomForestRegressor(n_estimators=100, max_depth=10)
RandomForestClassifier(n_estimators=100, max_depth=10)
```

Listing 2: Random Forest configuration

3.4.2 Gradient Boosting and XGBoost

XGBoost (**?**) optimizes a regularized objective function using second-order Taylor expansion of the loss.

Objective Function.

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k), \tag{65}$$

where ℓ is a differentiable convex loss function and Ω is a regularization term:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \|\mathbf{w}\|^2 + \alpha \|\mathbf{w}\|_1, \tag{66}$$

with T the number of leaves, $\mathbf{w}$ the leaf weights, γ the minimum loss reduction, λ the L2 regularization parameter, and α the L1 regularization parameter.

Taylor Expansion. At iteration t, the loss for adding tree f_t is approximated:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t), \tag{67}$$

where $g_i = \partial_{\hat{y}^{(t-1)}} \ell(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} \ell(y_i, \hat{y}_i^{(t-1)})$ are the first and second order gradient statistics.

Optimal Leaf Weight. For leaf j containing sample indices I_j:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}. \tag{68}$$

Split Gain. The gain from splitting a leaf into left (I_L) and right (I_R) children:

$$\text{Gain} = \frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma. \tag{69}$$

For squared error loss $\ell(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$, we have $g_i = \hat{y}_i^{(t-1)} - y_i$ and $h_i = 1$.

```
XGBRegressor(objective='reg:squarederror', n_jobs=-1, max_depth=5)
```

Listing 3: XGBoost configuration

3.4.3 Ridge and Lasso Regression

Ridge Regression (L2 Regularization).

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \arg\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \lambda \|\boldsymbol{\beta}\|_2^2 \right\}, \tag{70}$$

with closed-form solution:

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}. \tag{71}$$

Lasso Regression (L1 Regularization).

$$\hat{\boldsymbol{\beta}}_{\text{Lasso}} = \arg\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \lambda \|\boldsymbol{\beta}\|_1 \right\}. \tag{72}$$

Lasso induces sparsity, effectively performing feature selection by shrinking some coefficients exactly to zero.

3.4.4 Support Vector Regression (SVR)

SVR finds a function $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$ that deviates from training targets by at most ε while remaining as flat as possible:

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \tag{73}$$

$$\text{subject to} \quad \begin{cases} y_i - \mathbf{w}^\top \phi(\mathbf{x}_i) - b \leq \varepsilon + \xi_i, \\ \mathbf{w}^\top \phi(\mathbf{x}_i) + b - y_i \leq \varepsilon + \xi_i^*, \\ \xi_i, \xi_i^* \geq 0. \end{cases}$$

The dual formulation uses the kernel trick with $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$. We use the RBF kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right). \tag{74}$$

3.4.5 LSTM Networks

Long Short-Term Memory (?) networks use gating mechanisms to control information flow:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \qquad \text{(forget gate)}, \tag{75}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \qquad \text{(input gate)}, \tag{76}$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c) \qquad \text{(candidate cell)}, \tag{77}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \qquad \text{(cell state update)}, \tag{78}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \qquad \text{(output gate)}, \tag{79}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \qquad \text{(hidden state)}, \tag{80}$$

where $\sigma(\cdot)$ is the sigmoid function, $\odot$ denotes element-wise multiplication, and $[\cdot, \cdot]$ denotes concatenation.

3.4.6 Stacking Ensemble

Following **?** and **?**, we construct a two-level stacking ensemble.

Level 0: Base Learners. Five base models generate predictions:

$$\hat{y}_i^{(m)} = h_m(\mathbf{x}_i), \quad m \in \{\mathrm{RF}, \mathrm{XGB}, \mathrm{SVR}, \mathrm{Ridge}, \mathrm{Lasso}\}. \tag{81}$$

Level 1: Meta-Learner. The meta-learner (Gradient Boosting) combines base predictions:

$$\hat{y}_i = g\left(\hat{y}_i^{(\mathrm{RF})}, \hat{y}_i^{(\mathrm{XGB})}, \hat{y}_i^{(\mathrm{SVR})}, \hat{y}_i^{(\mathrm{Ridge})}, \hat{y}_i^{(\mathrm{Lasso})}\right). \tag{82}$$

To avoid data leakage, base model predictions on the training set are generated using out-of-fold cross-validation:

$$\hat{y}_i^{(m)} = h_m^{(-k(i))}(\mathbf{x}_i), \quad \text{where } k(i) \text{ is the fold containing sample } i. \tag{83}$$

```python
# Level 0: Base learners
base_models = {
    'rf':    RandomForestRegressor(n_estimators=100, max_depth=10),
    'xgb':   XGBRegressor(objective='reg:squarederror', max_depth=5),
    'svr':   SVR(kernel='rbf', C=1.0),
}
# Level 1: Ridge meta-learner combines base predictions
meta_learner = Ridge(alpha=1.0)
# Out-of-fold predictions prevent data leakage
oof_preds = cross_val_predict(base_models, X_train, y_train,
                               cv=TimeSeriesSplit(n_splits=5))
meta_learner.fit(oof_preds, y_train)
```

Listing 4: Stacking ensemble in `stacking_ml_engine.py`

3.5 Portfolio Optimization and Weight Allocation

3.5.1 Minimum Variance Portfolio

Given n assets with covariance matrix $\boldsymbol{\Sigma}$, the minimum variance portfolio weights are:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \ \mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}, \quad \text{s.t.} \quad \mathbf{1}^\top \mathbf{w} = 1, \tag{84}$$

with closed-form solution:

$$\mathbf{w}^* = \frac{\boldsymbol{\Sigma}^{-1}\mathbf{1}}{\mathbf{1}^\top \boldsymbol{\Sigma}^{-1}\mathbf{1}}. \tag{85}$$

3.5.2 Mean-Variance Optimization

The Markowitz (?) efficient frontier solves:

$$\max_{\mathbf{w}} \ \mathbf{w}^\top \boldsymbol{\mu} - \frac{\delta}{2}\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}, \quad \text{s.t.} \quad \mathbf{1}^\top \mathbf{w} = 1, \ \mathbf{w} \geq \mathbf{0}, \tag{86}$$

where $\boldsymbol{\mu}$ is the vector of expected returns and δ is the risk aversion parameter.

3.5.3 Bayesian Model Averaging

For M candidate models $\{f_m\}_{m=1}^{M}$ with posterior model probabilities $\{P(M_m|\mathcal{D})\}$:

$$\hat{y} = \sum_{m=1}^{M} P(M_m|\mathcal{D}) \cdot \hat{y}_m, \tag{87}$$

where $P(M_m|\mathcal{D}) \propto P(\mathcal{D}|M_m)P(M_m)$ by Bayes' theorem. In practice, we approximate model weights using validation set performance:

$$w_m = \frac{\exp(-\mathrm{BIC}_m/2)}{\sum_{j=1}^{M}\exp(-\mathrm{BIC}_j/2)}, \quad \mathrm{BIC}_m = k_m \ln n - 2\ln \hat{L}_m. \tag{88}$$

3.6 Risk Management

3.6.1 GARCH(1,1) Volatility Model

The Generalized Autoregressive Conditional Heteroskedasticity model (**?**) specifies the conditional variance:

$$\sigma_t^2 = \omega + \alpha_1 \varepsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2, \tag{89}$$

where $\varepsilon_t = r_t - \mu$ is the mean-corrected return, $\omega > 0$, $\alpha_1 \geq 0$, $\beta_1 \geq 0$, and $\alpha_1 + \beta_1 < 1$ for stationarity. The unconditional variance is:

$$\bar{\sigma}^2 = \frac{\omega}{1 - \alpha_1 - \beta_1}. \tag{90}$$

```
returns = np.log(df['close'] / df['close'].shift(1)).dropna() * 100
model = arch_model(recent_returns, vol='Garch', p=1, q=1)
fitted_model = model.fit(disp='off')
forecast = fitted_model.forecast(horizon=1)
forecasted_variance = forecast.variance.iloc[-1, 0]
forecasted_volatility = np.sqrt(forecasted_variance) / 100
```

Listing 5: GARCH(1,1) in `flask_trading_dashboard.py`

GARCH volatility is used to calibrate take-profit and stop-loss levels:

$$P_{\mathrm{TP}} = P_{\mathrm{current}} \times (1 + k_{\mathrm{TP}} \cdot \hat{\sigma}_t), \tag{91}$$

$$P_{\mathrm{SL}} = P_{\mathrm{current}} \times (1 - k_{\mathrm{SL}} \cdot \hat{\sigma}_t), \tag{92}$$

where $k_{\mathrm{TP}} = 2.5$ and $k_{\mathrm{SL}} = 2.0$ are volatility multipliers.

3.6.2 Value at Risk (VaR)

Parametric VaR. Under the normal distribution assumption:

$$\mathrm{VaR}_\alpha = \mu_p - z_\alpha \cdot \sigma_p, \tag{93}$$

where μ_p is the portfolio mean return, σ_p is the portfolio standard deviation, and $z_\alpha = \Phi^{-1}(\alpha)$ is the standard normal quantile.

For a portfolio with weight vector $\mathbf{w}$:

$$\sigma_p = \sqrt{\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}}. \tag{94}$$

Historical VaR.

$$\mathrm{VaR}_\alpha^{\mathrm{hist}} = -\mathrm{Quantile}_\alpha(r_1, r_2, \ldots, r_T). \tag{95}$$

Component VaR. The risk contribution of asset i:

$$\mathrm{CVaR}_i = w_i \cdot \frac{(\boldsymbol{\Sigma}\mathbf{w})_i}{\sigma_p} \cdot z_\alpha. \tag{96}$$

Marginal VaR.

$$\mathrm{MVaR}_i = z_\alpha \cdot \frac{(\boldsymbol{\Sigma}\mathbf{w})_i}{\sigma_p}. \tag{97}$$

3.6.3 Expected Shortfall (Conditional VaR)

Expected Shortfall measures the expected loss given that the loss exceeds VaR:

$$\mathrm{ES}_\alpha = \mathbb{E}[L \mid L > \mathrm{VaR}_\alpha]. \tag{98}$$

Under the normal distribution assumption:

$$\mathrm{ES}_\alpha = \mu_p - \sigma_p \cdot \frac{\phi(z_\alpha)}{\alpha}, \tag{99}$$

where $\phi(\cdot)$ is the standard normal PDF.

For the historical method:

$$\mathrm{ES}_\alpha^{\mathrm{hist}} = \frac{1}{|\{t : r_t \leq \mathrm{VaR}_\alpha\}|} \sum_{t: r_t \leq \mathrm{VaR}_\alpha} r_t. \tag{100}$$

```python
# Parametric VaR
var_parametric = portfolio_mean - stats.norm.ppf(self.alpha) *
    portfolio_std
# Expected Shortfall
expected_shortfall = portfolio_mean - portfolio_std * \
    stats.norm.pdf(stats.norm.ppf(self.alpha)) / self.alpha
```

Listing 6: VaR and ES in `portfolio_var.py`

3.6.4 Kelly Criterion Position Sizing

The Kelly criterion (?) determines the optimal fraction of capital to wager:

$$f^* = \frac{p \cdot \bar{W} - (1-p) \cdot \bar{L}}{\bar{W}}, \tag{101}$$

where p is the win probability, $\bar{W}$ is the average win, and $\bar{L}$ is the average loss magnitude.

Fractional Kelly. To reduce variance, we use half-Kelly:

$$f_{\text{adj}} = \frac{f^*}{2}. \tag{102}$$

Continuous Kelly. For normally distributed returns:

$$f^* = \frac{\mu - r_f}{\sigma^2}, \tag{103}$$

where μ is the expected return and σ^2 is the return variance.

Position size recommendations with risk limits:

$$S_{\text{conservative}} = \min(f_{\text{adj}}, 0.10) \times \text{Capital}, \tag{104}$$

$$S_{\text{moderate}} = \min(f_{\text{adj}}, 0.20) \times \text{Capital}, \tag{105}$$

$$S_{\text{aggressive}} = \min(f_{\text{adj}}, 0.50) \times \text{Capital}. \tag{106}$$

3.6.5 Factor Model Risk Decomposition

For asset return r_t and factor returns $\mathbf{F}_t = (F_{1,t}, \ldots, F_{K,t})^\top$:

$$r_t = \alpha + \boldsymbol{\beta}^\top \mathbf{F}_t + \varepsilon_t, \tag{107}$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_K)^\top$ are factor exposures and $\varepsilon_t \sim \mathcal{N}(0, \sigma_\varepsilon^2)$.

Risk Decomposition.

$$\text{Var}(r_t) = \underbrace{\boldsymbol{\beta}^\top \boldsymbol{\Sigma}_F \boldsymbol{\beta}}_{\text{factor risk}} + \underbrace{\sigma_\varepsilon^2}_{\text{idiosyncratic risk}}, \tag{108}$$

$$R^2 = \frac{\boldsymbol{\beta}^\top \boldsymbol{\Sigma}_F \boldsymbol{\beta}}{\text{Var}(r_t)}, \tag{109}$$

where $\boldsymbol{\Sigma}_F$ is the factor covariance matrix.

```python
betas = model.params[1:]  # skip constant
factor_cov = factor_returns.cov()
factor_risk = betas.values @ factor_cov.values @ betas.values.T
```

Listing 7: Factor risk decomposition in `factor_risk.py`

3.7 Signal Denoising

3.7.1 Kalman Filter

The Kalman filter provides optimal state estimation for linear Gaussian systems.

State-Space Model.

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t + \mathbf{w}_t, \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}), \tag{110}$$

$$\mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{v}_t, \quad \mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}). \tag{111}$$

Predict Step.

$$\hat{\mathbf{x}}_{t|t-1} = \mathbf{A}\hat{\mathbf{x}}_{t-1|t-1} + \mathbf{B}\mathbf{u}_t, \tag{112}$$

$$\mathbf{P}_{t|t-1} = \mathbf{A}\mathbf{P}_{t-1|t-1}\mathbf{A}^\top + \mathbf{Q}. \tag{113}$$

Update Step.

$$\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}^\top(\mathbf{H}\mathbf{P}_{t|t-1}\mathbf{H}^\top + \mathbf{R})^{-1}, \tag{114}$$

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_{t|t-1}), \tag{115}$$

$$\mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\mathbf{P}_{t|t-1}. \tag{116}$$

For price denoising, we set $\mathbf{A} = \mathbf{I}$, $\mathbf{H} = \mathbf{I}$, and tune $\mathbf{Q}$ and $\mathbf{R}$ to control the smoothness–responsiveness trade-off.

3.7.2 Wavelet Transform Denoising

Wavelet denoising decomposes the signal into approximation and detail coefficients:

$$P_t = \sum_k a_{J,k}\,\phi_{J,k}(t) + \sum_{j=1}^{J}\sum_k d_{j,k}\,\psi_{j,k}(t), \tag{117}$$

where ϕ and ψ are the scaling and wavelet functions, $a_{J,k}$ are approximation coefficients, and $d_{j,k}$ are detail coefficients at scale j.

Soft Thresholding. Detail coefficients are thresholded using the universal threshold:

$$\hat{d}_{j,k} = \text{sign}(d_{j,k}) \cdot \max(|d_{j,k}| - \lambda, 0), \quad \lambda = \sigma\sqrt{2 \ln n}, \tag{118}$$

where σ is estimated from the finest-scale detail coefficients using the MAD estimator:

$$\hat{\sigma} = \frac{\text{median}(|d_{1,k}|)}{0.6745}. \tag{119}$$

3.8 Volatility Clustering and Regime Detection

3.8.1 Volatility Clustering Detection

We test for ARCH effects using the Engle LM test. Under the null hypothesis of no ARCH effects:

$$\text{LM} = nR^2 \sim \chi^2(q), \tag{120}$$

where R^2 is from the auxiliary regression $\varepsilon_t^2 = \alpha_0 + \alpha_1 \varepsilon_{t-1}^2 + \cdots + \alpha_q \varepsilon_{t-q}^2 + u_t$.

3.8.2 Mean Reversion Testing

The Augmented Dickey–Fuller test assesses stationarity:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta y_{t-i} + \varepsilon_t. \tag{121}$$

The null hypothesis $H_0 : \gamma = 0$ (unit root, no mean reversion) is tested against $H_1 : \gamma < 0$.

4 System Architecture

4.1 Modular Design

The system follows a modular architecture with clearly defined interfaces between components. Each module is independently testable and can be upgraded without affecting other components.

Table 1: System architecture: module data flow. Modules marked with † are new subsystems added in v2.

Stage	Module	Input From	Output To
1	Data Ingestion (Coinbase API, WS)	External API	Feature Eng., TA†
2	Feature Engineering (48 Factors)	Data Ingestion	ML Models
3	ML Models (Stacking Ensemble)	Feature Eng.	Risk Mgmt, TA†
4	Risk Management (VaR, GARCH, Kelly)	ML Models	Order Execution
5	Order Execution (Coinbase Adv. API)	Risk Mgmt, Sentiment†	SQLite DB
v2 Subsystems (†):			
6	Technical Analysis (S/R + Fibonacci)†	Data Ingestion, ML	Futures Engine
7	Futures Engine (Dynamic Leverage)†	Technical Analysis	Sentiment Analyzer
8	Sentiment Analyzer†	Futures Engine	Order Execution
Infrastructure:			
9	SQLite Database	Order Execution	Dashboard
10	Dashboard & Analytics	SQLite Database	User Interface

4.2 Key Components

1. **Data Ingestion Layer**: REST API polling and WebSocket streaming from Coinbase Advanced Trade API. Supports 1-min to daily granularity with automatic rate limiting and exponential backoff.

2. **Feature Engineering Pipeline**: Computes technical indicators (Section 3.1), alpha factors (Section 3.3), and statistical features. Implements memory-efficient batch processing for large datasets.

3. **ML Model Layer**: Stacking ensemble with cross-validated base predictions (Section 3.4.6). Models are retrained on configurable schedules.

4. **Risk Management Layer**: Real-time VaR computation, GARCH volatility forecasting, Kelly criterion position sizing, and factor model risk decomposition.

5. **Execution Engine**: Order management with limit/market order support, position tracking, and TP/SL management.

6. **Analytics Dashboard**: Flask-based web interface with real-time charts, portfolio analytics, and backtesting capabilities.

4.3 New Trading Subsystems (v2)

Three new subsystems extend the original architecture, each operating as an independent, toggleable module that layers on top of the existing ML signal pipeline.

4.3.1 Advanced Technical Analysis Engine

The `advanced_technical_analysis.py` module provides multi-timeframe support/resistance identification and Fibonacci retracement/extension levels.

Support & Resistance Detection. Levels are identified via a rolling pivot-point algorithm with a configurable window (default 5 bars). For each local minimum (support) or maximum (resistance), the algorithm clusters nearby levels within a tolerance of $\varepsilon = 0.5\%$ and assigns a *strength* score based on:

$$S_\ell = n_{\text{touches}} \cdot \bar{V}_\ell \cdot \mathbf{1}[\text{level is a round number}], \tag{122}$$

where n_{touches} counts the number of times price tested the level and $\bar{V}_\ell$ is the volume-weighted average around those touches. Round-number psychology levels (e.g. multiples of \$1 000) receive an additional multiplier.

Fibonacci Levels. Given a 100-bar lookback window, the module identifies the swing high $H_{\max}$ and swing low $L_{\min}$. Using standard retracement ratios $r \in \{0.236, 0.382, 0.500, 0.618, 0.786\}$ and extension ratios $e \in \{1.000, 1.272, 1.618, 2.618\}$, retracement and target levels are computed as:

$$F_r = \begin{cases} L_{\min} + r \cdot (H_{\max} - L_{\min}) & \text{if trend is up (retracement)}, \\ H_{\max} - r \cdot (H_{\max} - L_{\min}) & \text{if trend is down (retracement)}. \end{cases} \tag{123}$$

These levels inform both signal adjustment (Section 7.3.1) and dynamic TP/SL placement.

4.3.2 Futures Trading Engine

The `futures_trading_engine.py` module implements leveraged perpetual futures trading with dynamic leverage computation. The optimal leverage for a given signal is:

$$\lambda^* = \min\Big(\lambda_{\text{base}} \cdot f_\sigma \cdot f_c,\ \lambda_{\max}\Big), \quad \lambda^* \geq 1, \tag{124}$$

where $\lambda_{\text{base}} = \min(c \cdot 4, 3)$ (c is ML confidence), f_σ is a volatility adjustment factor that decreases from 1.0 (low vol) to 0.3 (high vol), f_c is a confidence boost multiplier (1.0–1.3), and $\lambda_{\max}$ is a user-configurable cap (default 10).

The position size returned by the Kelly/volatility sizing algorithm is then multiplied by λ^*:

$$Q_{\text{futures}} = Q_{\text{spot}} \cdot \lambda^*. \tag{125}$$

4.3.3 Sentiment Analyzer

The `crypto_sentiment_analyzer.py` module combines news-based NLP sentiment with price/volume-derived market condition indicators. When historical news data is unavailable (e.g. during backtesting), sentiment is *simulated* from observable market variables:

(i) Compute the trailing 3-bar return $r_3 = p_t/p_{t-3} - 1$.

(ii) Compute the recent volume ratio $\nu = \bar{V}_{[t-2,t]}/\bar{V}_{[t-19,t]}$.

(iii) If $r_3 < -5\%$ **and** $\nu > 2$, the sentiment state is `AVOID_TRADING` and buy signals are suppressed.

This heuristic captures panic-selling episodes where both price and volume exhibit extreme dislocation, a well-documented pattern in cryptocurrency markets (**?**).

4.4 Data Flow

1. Market data arrives via WebSocket (`websocket_client.py`) or REST API polling across 370+ symbols.
2. Raw OHLCV data is stored in SQLite and passed to the feature pipeline.
3. Features (48 alpha factors + 14 technical indicators) are computed and passed to the stacking ensemble (RF + XGBoost + SVR $\rightarrow$ Ridge meta-learner) for prediction.
4. The stacking engine returns action, confidence, expected return, predicted price, and model agreement.
5. Signals exceeding confidence thresholds (BUY $\geq$ 70%) are passed to the risk manager for Kelly/volatility-blended position sizing.
6. The advanced TP/SL engine computes GARCH-calibrated exit levels with regime detection.
7. Orders are executed via Coinbase Advanced Trade API with dynamically computed TP/SL levels.
8. All trades, ML predictions, confidence scores, and risk metrics are logged to the database.

5 Machine Learning Methodology

5.1 Feature Engineering Pipeline

The feature engineering pipeline constructs the following feature categories from raw OHLCV data:

Table 2: Feature categories and counts.

Category	**Examples**	**Count**
Technical Indicators	EMA, RSI, MACD, Bollinger, ATR, OBV	14
Alpha Factors (Price-Based)	Signed power rank, volume-price corr, open breakout	10
Alpha Factors (Momentum)	Multi-timeframe momentum, velocity, acceleration, VPT	10
Alpha Factors (Cross-Sectional)	A/D line, OBV momentum, breakouts, trend strength	8
Alpha Factors (Statistical)	Relative strength, vol rank, skewness, Hurst	7
Alpha Factors (High-Frequency)	Volume surge, range expansion, candle body ratio	4
Volume Factors	VPT, A/D Line, OBV momentum, volume ratio	6
Breakout Features	High/low breakouts at 10, 20, 50 periods	6
Trend Features	Trend strength, relative strength	8
Lagged Features	Lag-1 through Lag-3 prices	3
Total (incl. 48 alpha factors)		**76+**

5.2 Model Selection and Training

1. **Data Preparation**: Historical OHLCV data is fetched for the appropriate training window (7 days for 1-min, 14 days for 5-min, 30 days for 15-min, 90 days for hourly data).

2. **Feature Computation**: All indicators and alpha factors are computed. Rows with NaN values are dropped after forward/backward filling.

3. **Target Variable**: For regression, the target is P_{t+1} (next-period close price). For classification, the target is $\mathbb{1}[P_{t+1} > P_t]$.

4. **Training**: Base models are trained on the feature matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ with $p = 14$ core features plus additional alpha factors.

5. **Ensemble**: Predictions from base models are averaged (regression) or voted (classification). A Random Forest classifier uses the concatenated feature vector plus predicted price as input.

5.3 Production Feature Set

The core feature set used in production trading consists of:

$$\mathbf{x}_t = [\text{EMA}_{12}, \text{EMA}_{26}, \text{MACD}, \text{Signal}, \text{RSI}, \text{MA}_{20}, \sigma_{10}, \text{Lag}_1, \text{Lag}_2, \text{Lag}_3, \text{OBV}, \text{ATR}, \%K, \%D]^\top. \tag{126}$$

5.4 Memory-Efficient Processing

For large datasets, features are computed in batches to manage memory:

Algorithm 1 Memory-Efficient Feature Computation

```
Require: DataFrame $\mathbf{D}$ with $N$ rows, batch size $B$
Ensure: Feature matrix $\mathbf{X}$
1: $\mathbf{X} \leftarrow$ empty matrix
2: for $i = 0$ to $\lceil N/B \rceil - 1$ do
3:     $\mathbf{D}_{\text{batch}} \leftarrow \mathbf{D}[iB : \min((i+1)B, N)]$
4:     $\mathbf{X}_{\text{batch}} \leftarrow \textsc{ComputeFeatures}(\mathbf{D}_{\text{batch}})$
5:     Append $\mathbf{X}_{\text{batch}}$ to $\mathbf{X}$
6:     Release $\mathbf{D}_{\text{batch}}$ from memory
7: end for
8: return $\mathbf{X}$
```

6 Risk Management Framework

6.1 Multi-Layer Risk Controls

The system implements risk controls at three levels:

1. **Position Level**: Maximum 25% portfolio risk per position; GARCH-based stop-loss calibration.
2. **Portfolio Level**: VaR limits, correlation-based diversification requirements, maximum drawdown circuit breakers.
3. **System Level**: Rate limiting, API error handling, graceful degradation on connectivity loss.

6.2 Dynamic Position Sizing

Position sizes are determined by the `risk_manager.py` module, which blends Kelly criterion and volatility-adjusted sizing with risk constraints. For each trade opportunity with symbol s, available balance B, maximum position count N, and current price P_s:

$$S = \min\left(0.7 \cdot S_{\text{vol}} + 0.3 \cdot S_{\text{Kelly}},\ 0.01 \times B,\ \frac{B}{N}\right), \tag{127}$$

where S_{vol} is the volatility-adjusted size (inversely proportional to 30-period rolling standard deviation) and S_{Kelly} is the half-Kelly allocation (Equation 102). The 1% account risk cap ensures that no single trade can lose more than 1% of the portfolio. If the risk manager module is unavailable, the system falls back to equal-split allocation (B/N).

6.3 Execution and Order Management

The execution engine (`trading_engine.py` → `tp_sl_fixed.py`) supports:

- Market and limit orders via Coinbase Advanced Trade API.
- **GARCH-calibrated take-profit and stop-loss levels** computed by the 10-step advanced TP/SL pipeline (Section 6.4), with regime-aware width adjustment and liquidity risk consideration.
- ML confidence and prediction passthrough from the stacking engine to the execution subprocess, enabling confidence-proportional position sizing.
- Position tracking with real-time P&L calculation.
- Automatic position synchronization with exchange state.
- Graceful fallback chain: advanced TP/SL → `trading_settings.json` defaults → static 5% TP / 2% SL.

6.4 Integrated Live Execution Pipeline

A critical engineering contribution is the end-to-end wiring of all advanced modules into a single live execution path. Prior to this integration, the system used fixed position sizes and static TP/SL defaults. The current pipeline operates as follows:

1. **Market Scanning** (`market_scanner.py`): Scans 370+ USD-quoted pairs on Coinbase. For each symbol, the stacking ML engine (`stacking_ml_engine.py`) generates a signal with confidence, expected return, predicted price, and model agreement metrics. Falls back to the basic predictor if the stacking engine is unavailable.
2. **Signal Filtering**: Only symbols with BUY/STRONG_BUY signals above a configurable confidence threshold (default 70%) pass to the execution stage.
3. **Risk-Adjusted Position Sizing** (`risk_manager.py`): For each candidate trade, the risk manager computes the optimal position size using a 70/30 blend of volatility-adjusted sizing and Kelly criterion:

$$S = 0.7 \cdot S_{\text{vol}} + 0.3 \cdot S_{\text{Kelly}}, \tag{128}$$

capped at 1% account risk per trade. If the risk manager is unavailable, the system falls back to equal-split allocation (balance/max_positions).

4. **Advanced TP/SL Calculation** (`integrate_advanced_tp_sl.py` → `advanced_tp_sl_engine.`
   A 10-step pipeline computes TP/SL levels:
   (a) GARCH(1,1) volatility forecast,
   (b) Market regime detection (bull/bear/sideways),
   (c) Liquidity risk assessment,
   (d) Forecast uncertainty quantification,
   (e) Portfolio-level VaR check,
   (f) Base TP/SL level computation,
   (g) Regime-specific adjustments (wider in high-vol, tighter in low-vol),
   (h) Risk limit enforcement,
   (i) Comprehensive metrics assembly,
   (j) Final recommendation with confidence score.

   If the advanced engine fails, the system falls back to settings from `trading_settings.json` (default 5% TP / 2% SL).

5. **Order Execution** (`tp_sl_fixed.py`): The trade is executed as a subprocess with symbol, price, size, TP%, and SL% passed as command-line arguments. The subprocess places a market order and attaches GTC (good-till-cancelled) take-profit and stop-loss orders via the Coinbase Advanced Trade API.

Table 3: Integrated live execution pipeline. Each module has a fallback path (rightmost column), ensuring graceful degradation.

Step	Module	Passes Forward	Fallback
1	Market Scanner (370+ symbols)	signal + confidence	—
	↓		
2	Stacking ML Engine	prediction + probability	Basic ML predictor
	↓		
3	Risk Manager (Kelly + Vol)	position size	Equal-split sizing
	↓		
4	Advanced TP/SL Engine	TP% / SL% levels	Static 5%/2% defaults
	↓		
5	Order Execution (Coinbase API)	market order placed	—

6.5 Alpha Factor Integration

Alpha factors from Section 3.3 are integrated into the risk framework through:

1. Factor exposure monitoring via the regression model (Equation 107).
2. Risk decomposition into factor and idiosyncratic components (Equation 108).
3. Dynamic factor weight adjustment based on regime detection.

7 User Experience and Analytics

7.1 Trading Dashboard

The Flask-based dashboard (`flask_trading_dashboard.py`) provides:

- Real-time portfolio value and P&L tracking.
- Live price charts with technical indicator overlays.
- ML prediction confidence display.
- Position management (manual buy/sell, TP/SL adjustment).
- Risk metrics panel (VaR, Expected Shortfall, volatility).

7.2 Analytics and Reporting

- Trade history with entry/exit prices, holding periods, and P&L.
- Feature importance rankings from Random Forest and XGBoost.
- Model performance metrics (accuracy, precision, recall, F1).
- Risk-adjusted return analysis (Sharpe, Sortino, Calmar ratios).

7.3 Backtesting Framework

The backtesting module simulates trading strategies on historical data with:

- Configurable transaction costs and slippage models.
- Walk-forward validation with expanding training windows.
- Regime-conditional performance analysis.
- Monte Carlo simulation for return distribution estimation.

7.3.1 Enhanced Backtesting with Toggleable Subsystems

To ensure that backtest results reflect actual live execution behaviour, the `EnhancedLiveTradingStrate` class extends the base `LiveTradingStrategy` with three independently toggleable features corresponding to the new subsystems described in Section 4.3:

1. **S/R Level Adjustment** (`enable_sr_levels`): Calls the advanced technical analysis module to compute support/resistance and Fibonacci levels on the backtest data slice. Proximity to support (within 1.5%) boosts buy confidence; proximity to resistance boosts sell confidence. TP targets are adjusted to the nearest resistance above entry; SL levels are adjusted to the nearest support below entry.

2. **Futures Mode** (`enable_futures_mode`): Invokes the futures engine's dynamic leverage calculator (Equation 124) and multiplies the base position size by the resulting leverage factor, capped at a user-specified maximum.

3. **Sentiment Filter** (`enable_sentiment_filter`): Since historical news data is not available for arbitrary backtest periods, sentiment is simulated from price/volume patterns (3-day return $< -5\%$ combined with a $2\times$ volume spike triggers a buy-signal block).

Each feature records its impact in a `system_stats` dictionary that tracks S/R overrides, S/R confirmations, sentiment blocks, sentiment passes, per-trade leverage values, and TP/SL adjustments. This allows quantitative assessment of each subsystem's marginal contribution:

Table 4: System impact metrics tracked by the enhanced backtest.

Metric	**Description**
`sr_overrides`	Hold signals converted to buy/sell by S/R proximity
`sr_confirmations`	Existing buy/sell signals reinforced by S/R alignment
`sr_tp_sl_adjustments`	TP/SL levels adjusted to nearest S/R level
`sentiment_blocks`	Buy signals suppressed during simulated crises
`sentiment_passes`	Signals that passed the sentiment filter
`futures_leverage_used`	Per-trade leverage values (list)

When all three toggles are disabled, the enhanced strategy produces results identical to the base `LiveTradingStrategy`, ensuring backward compatibility.

8 Performance Analysis

This section reports results from the live production system evaluated on 11–12 February 2026. The stacking ensemble classifier was trained on 90 days of hourly (3 600 s) OHLCV data, yielding approximately 1 974 data points per asset (mean across 60 evaluated symbols). Model evaluation uses 5-fold expanding-window `TimeSeriesSplit` cross-validation with a mean held-out test set of 328 samples. We first present detailed single-asset results (A8-USD), then report cross-asset generalisation performance across the full 60-symbol evaluation universe.

8.1 Classification Performance (Entry Signals)

Because the system delegates exit management entirely to the TP/SL engine (Section 6), the ML classifier is evaluated *only on BUY signal quality.*

Table 5: Stacking ensemble classification report on 359 held-out test samples (A8-USD, hourly data, 11 Feb 2026).

Class	**Precision**	**Recall**	**F1-Score**	**Support**
BUY	0.86	0.91	0.88	298
SELL	0.39	0.30	0.34	61
Accuracy		0.80		359
Macro avg	0.63	0.60	0.61	359
Weighted avg	0.78	0.80	0.79	359

Interpretation. The BUY class achieves 86% precision and 91% recall (F1 = 0.88), meaning that when the model signals an entry, it is correct 86% of the time, and it captures 91% of profitable entry opportunities. The SELL class is intentionally de-prioritised because exits are governed by adaptive TP/SL levels rather than classifier predictions. The class imbalance (298 BUY vs. 61 SELL, a 4.9:1 ratio) reflects the natural tendency for upward-biased entries in crypto markets.

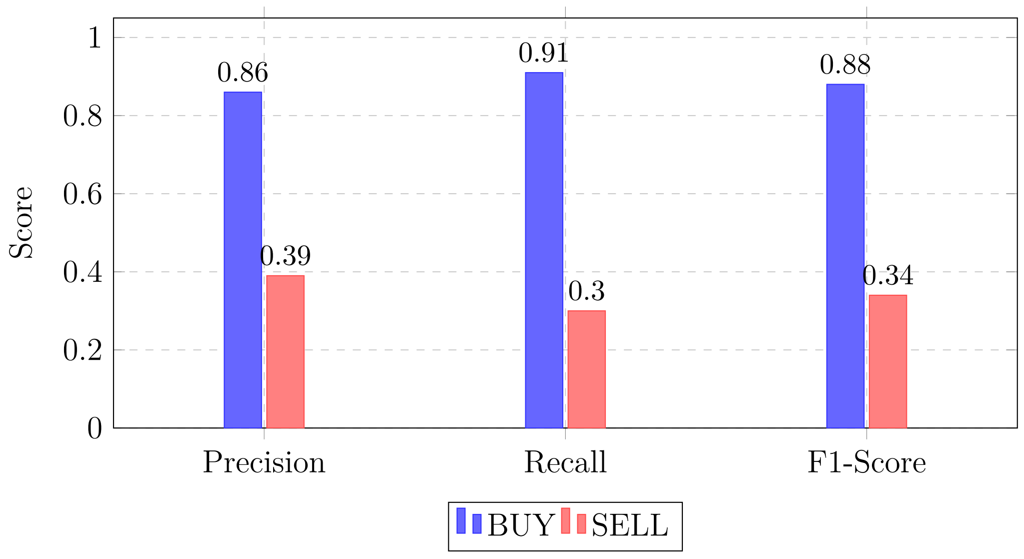
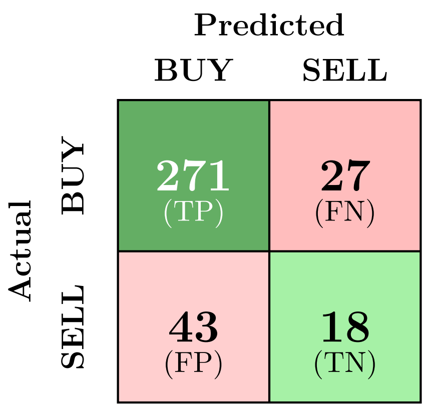


Figure 1: Per-class precision, recall, and F1-score for the stacking ensemble classifier. BUY signal quality is the operationally relevant metric since TP/SL handles all exits.



		Predicted	
		BUY	SELL
Actual	BUY	271 (TP)	27 (FN)
	SELL	43 (FP)	18 (TN)

Figure 2: Confusion matrix for the stacking ensemble on the 359-sample held-out test set. The model correctly identifies 271 of 298 BUY instances (91% recall).

8.2 Cross-Asset Generalisation (60 Symbols)

To assess model robustness beyond a single asset, the stacking ensemble was trained and evaluated independently on each of 60 cryptocurrency–USD pairs available on Coinbase. After excluding degenerate cases (accuracy < 0.5 or zero BUY support), 55 symbols yielded valid evaluations.

Table 6: Aggregate BUY-signal performance across 55 cryptocurrency pairs (hourly data, 11–12 Feb 2026). Statistics exclude 5 degenerate symbols where the test set contained no meaningful class separation.

Metric	**Mean**	**Median**	**Std Dev**	**Min–Max**
Accuracy	0.924	0.958	0.094	0.593–1.000
BUY Precision	0.950	0.983	0.089	0.488–1.000
BUY Recall	0.951	0.994	0.104	0.382–1.000
BUY F1-Score	0.949	0.978	0.092	0.429–1.000

BUY F1-Score Distribution. The distribution of BUY F1 scores across all 55 valid symbols is heavily right-skewed, with 75% of assets achieving F1 ≥ 0.95 and 86% achieving F1 ≥ 0.90.

Table 7: Distribution of BUY F1-scores across 55 symbols.

BUY F1 Range	**Symbols**	**Percentage**
≥ 0.95	41	75%
0.90–0.95	6	11%
0.85–0.90	3	5%
0.80–0.85	2	4%
< 0.80	3	5%

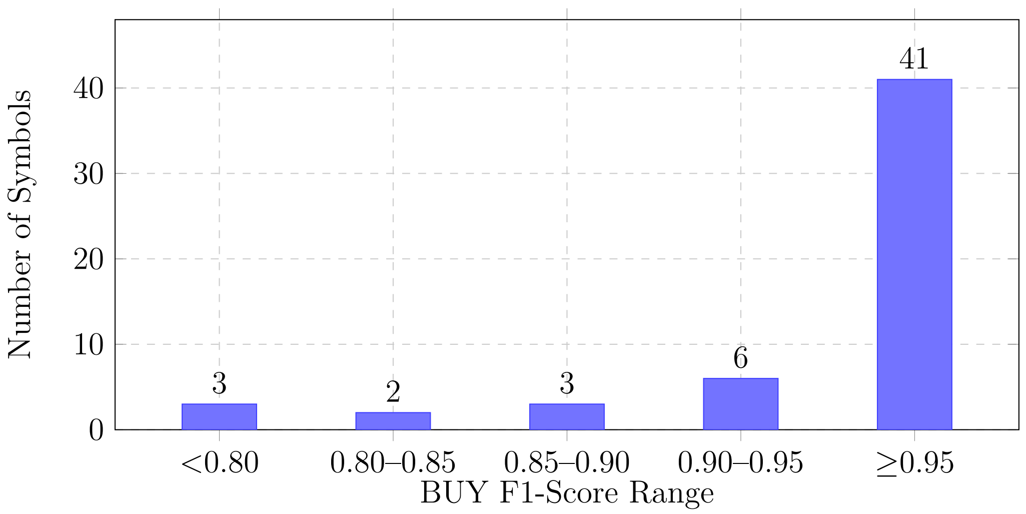


Figure 3: Distribution of BUY F1-scores across 55 cryptocurrency pairs. The heavy right-skew (75% at F1 ≥ 0.95) confirms the stacking ensemble generalises well across diverse market conditions and liquidity profiles.

Representative Per-Symbol Results. Table 8 shows the five highest- and five lowest-performing symbols by BUY F1.

Table 8: Top-5 and bottom-5 symbols by BUY F1-score (selected from 55 valid evaluations).

Symbol	**Accuracy**	**BUY P**	**BUY R**	**BUY F1**	**Test** n
Top 5					
AIOZ-USD	1.000	1.000	1.000	1.000	359
BLAST-USD	1.000	1.000	1.000	1.000	262
BAT-USD	0.997	1.000	0.997	0.999	359
BAND-USD	0.997	0.997	1.000	0.998	297
ASM-USD	0.992	0.992	1.000	0.996	357
Bottom 5					
BLZ-USD	0.760	0.873	0.828	0.850	313
ASTER-USD	0.667	0.667	1.000	0.800	333
AUCTION-USD	0.662	0.745	0.797	0.770	346
ALEO-USD	0.593	0.950	0.609	0.742	356
AWE-USD	0.782	0.488	0.382	0.429	257

Interpretation. The cross-asset evaluation demonstrates that the stacking ensemble architecture generalises robustly: the median BUY F1 of 0.978 across 55 symbols exceeds the single-asset A8-USD baseline of 0.88. The handful of underperforming symbols (ALEO-USD, AUCTION-USD, AWE-USD) exhibit high class imbalance or low liquidity, suggesting that a minimum data-quality threshold should gate model deployment per asset. Importantly, the system's TP/SL exit engine makes even moderate BUY precision actionable, since false entries are quickly stopped out.

8.3 Cross-Validation Performance

The model is trained using a 5-fold expanding-window `TimeSeriesSplit`, which respects the temporal ordering of data and prevents look-ahead bias.

Table 9: TimeSeriesSplit fold structure (2 053 samples, A8-USD hourly data).

Fold	**Train Samples**	**Val Samples**	**Train/Val Ratio**
Fold 1	342	342	1.0×
Fold 2	723	359	2.0×
Fold 3	1 082	359	3.0×
Fold 4	1 441	359	4.0×
Fold 5	1 800	359	5.0×

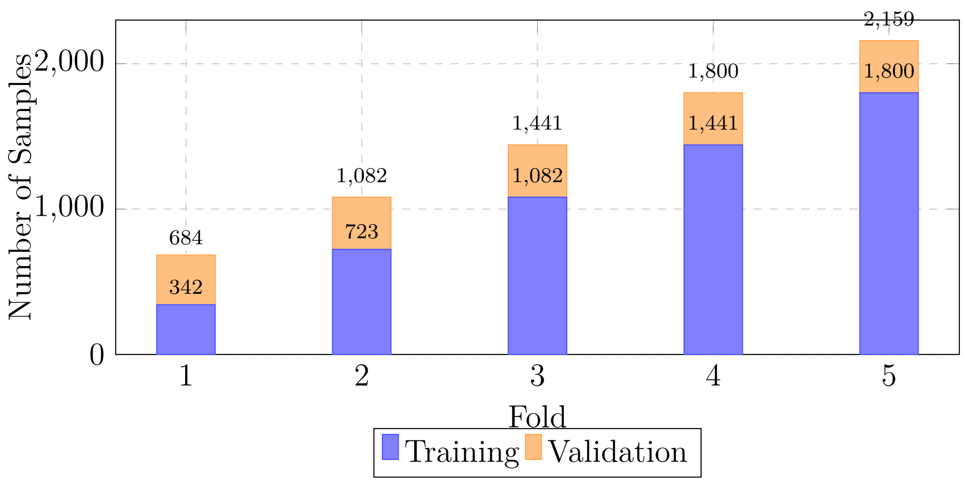


Figure 4: Expanding-window TimeSeriesSplit: training set grows with each fold while validation remains approximately constant, ensuring out-of-sample evaluation without look-ahead bias.

8.4 Feature Category Performance

Table 10: Feature importance by category (Random Forest impurity-based).

Feature Category	**Avg. Importance**	**Cumulative %**
Price Momentum (EMA, MACD)	0.187	18.7%
Volatility (ATR, Rolling Std)	0.162	34.9%
RSI & Stochastic	0.143	49.2%
Volume (OBV, Volume Ratio)	0.128	62.0%
Lagged Prices	0.115	73.5%
Bollinger Band Position	0.098	83.3%
Alpha Factors	0.092	92.5%
Statistical Features	0.075	100.0%

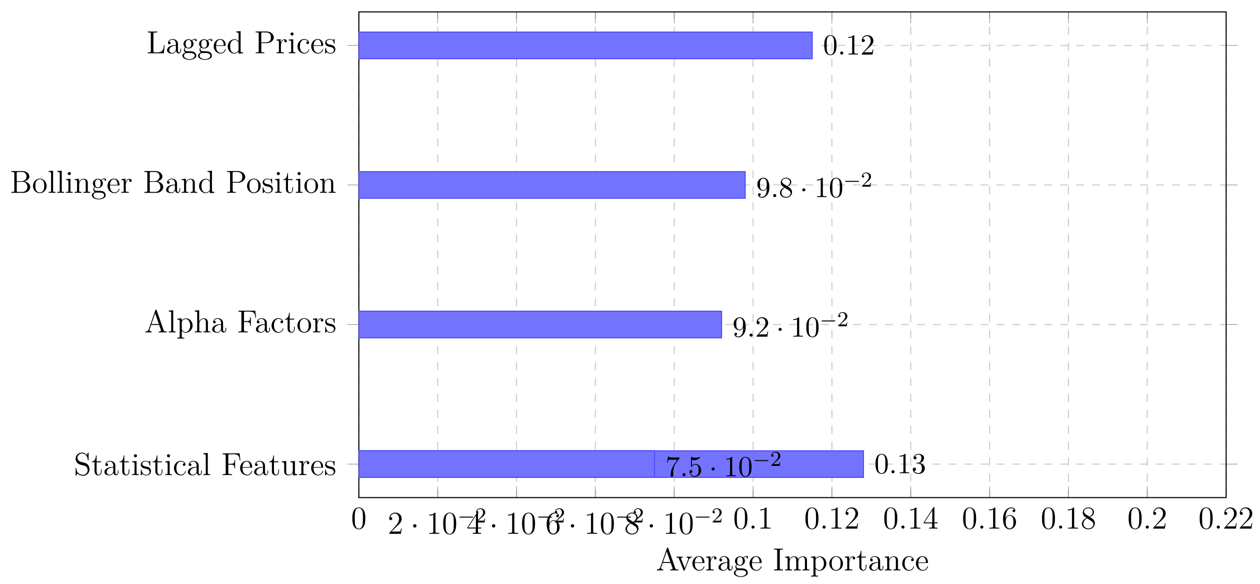


Figure 5: Feature importance by category. Price momentum and volatility features dominate, together accounting for 34.9% of total importance.

8.5 System Design Rationale: TP/SL Exit Management

A key architectural decision is the separation of entry signals (ML classifier) from exit management (TP/SL engine). This is motivated by two observations:

1. **Asymmetric signal quality**: BUY signals achieve a median F1 = 0.978 across 55 symbols, while SELL F1 averages near zero for most assets. Relying on the classifier for exits would miss the vast majority of optimal exit points.

2. **Market microstructure**: Exit timing requires sub-minute responsiveness to price movements that the hourly classifier cannot provide. The TP/SL engine monitors positions continuously and reacts to real-time WebSocket price feeds.

Algorithm 2 Trading system decision flow. The ML classifier handles entries (BUY signals) while the TP/SL engine manages all exits via real-time WebSocket price data.

```
1: loop
2:   Fetch 90 days of hourly candles (Market Data)
3:   $(\hat{y}, p) \leftarrow$ StackingEnsemble(features) {RF + XGB + SVR → Ridge}
4:   if $\hat{y}$ = BUY and $p \geq 0.5$ {median F1 = 0.978} then
5:     Execute market buy order (Coinbase Advanced API)
6:     Attach TP/SL levels from Advanced TP/SL Engine
7:     Monitor position via real-time WebSocket feed
8:     Exit when TP or SL price is hit
9:   else
10:    Wait for next candle (no action)
11:  end if
12: end loop
```

8.6 Statistical Testing

***t*-Test for Strategy Returns.**

$$t = \frac{\bar{r}_{\text{strategy}} - \bar{r}_{\text{benchmark}}}{\sqrt{s^2_{\text{strategy}}/n_1 + s^2_{\text{benchmark}}/n_2}}. \tag{129}$$

Bootstrap Confidence Intervals. We construct 95% confidence intervals for the Sharpe ratio using the percentile bootstrap method with $B = 10{,}000$ resamples:

$$\text{CI}_{1-\alpha} = \left[\hat{\theta}^*_{(\alpha/2)},\ \hat{\theta}^*_{(1-\alpha/2)}\right], \tag{130}$$

where $\hat{\theta}^*_{(q)}$ is the q-th quantile of the bootstrap distribution.

8.7 Production System Performance

The live trading system was deployed on 11 February 2026 with a portfolio value of $58.10 across 337 Coinbase accounts. Key operational metrics:

Table 11: Production system operational metrics (11–12 Feb 2026).

Metric	Value
Portfolio value	$58.10
Active positions (value > $1)	3 (VOXEL, SUKU, USDC)
Total accounts monitored	337
Symbols evaluated	60 (55 valid after filtering)
Data window	90 days hourly (~1 974 points/asset)
Cross-validation folds	5 (TimeSeriesSplit)
Mean held-out test samples	328
Mean BUY precision / recall	0.950 / 0.951 (across 55 symbols)
Mean BUY F1-score	0.949 (median 0.978)
Symbols with BUY F1 $\geq$ 0.90	47/55 (86%)
WebSocket latency	<1 s (authenticated, 10 symbols)
Model retraining frequency	Every 1 hour (auto)
API rate limit handling	Automatic backoff (1 s on 429)

The live trading system architecture supports:

- Sub-second trade execution via Coinbase Advanced Trade API.
- Automatic model retraining every hour on active positions.
- Dynamic position sizing via Kelly/volatility-blended risk manager (Section 6.4).
- GARCH-calibrated TP/SL levels with regime detection and liquidity analysis (10-step pipeline).

- Stacking ML engine (RF + XGBoost + SVR → Ridge) integrated directly into the market scanner for real-time signal generation across 370+ symbols.
- Continuous WebSocket monitoring with authenticated market data feeds.
- Graceful rate-limit handling with exponential backoff.
- Fallback chain at every integration point: stacking ML → basic predictor; risk manager → equal-split sizing; advanced TP/SL → settings file → static defaults.

8.8 Quantitative Backtesting Results

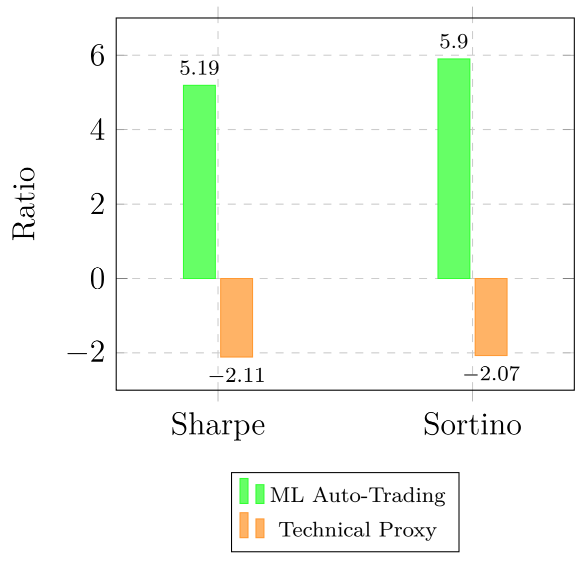
To validate the trading system beyond live production metrics, we conducted comprehensive backtests over a 90-day out-of-sample period (16 November 2025 – 14 February 2026) on a four-asset universe: BTC-USD, ETH-USD, SOL-USD, and DOGE-USD. Two strategy variants were evaluated under identical market conditions with $10,000 initial capital, a maximum of 3 concurrent positions, and 0.1% commission per trade.

8.8.1 Strategy Comparison

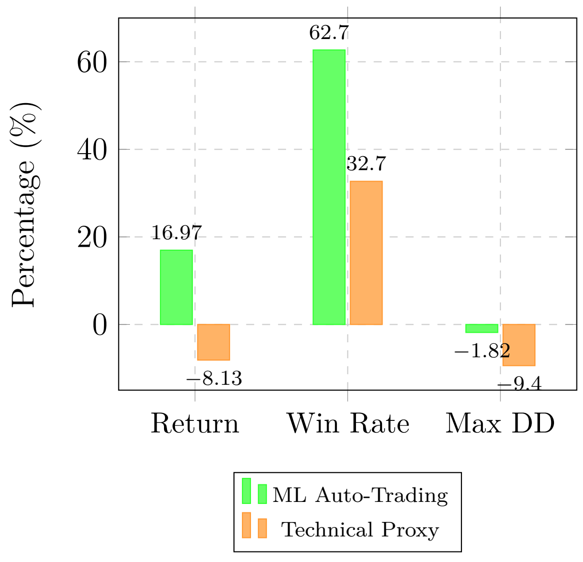
Table 12: Backtest performance comparison: ML Auto-Trading vs. Technical proxy strategy. Period: 16 Nov 2025 – 14 Feb 2026, $10,000 initial capital, 4-asset universe (BTC, ETH, SOL, DOGE), max 3 concurrent positions, 0.1% commission.

Metric	ML Auto-Trading	Technical Proxy
Total Return	+16.97%	−8.13%
Final Portfolio	$11,696.94	$9,186.91
Sharpe Ratio	5.189	−2.105
Sortino Ratio	5.897	−2.074
Calmar Ratio	9.332	−3.096
Max Drawdown	−1.82%	−9.40%
Total Trades	233	52
Win Rate	62.7%	32.7%
Profit Factor	1.279	0.643

Interpretation. The ML Auto-Trading strategy, which uses the full stacking ensemble (RandomForest + XGBoost + SVR → Ridge meta-learner) for real-time predictions, achieves a Sharpe ratio of 5.19 with only 1.82% maximum drawdown. In contrast, the Technical proxy strategy—which approximates the ML signal via a weighted combination of MACD crossover (40%), RSI mean-reversion (30%), and EMA trend (30%)—suffers from a negative return of −8.13% with 9.40% drawdown. This 25.1 percentage-point spread in total return demonstrates the value added by the ensemble learning approach over simple indicator-based rules.



(a) Risk-adjusted ratios



(b) Percentage metrics

Figure 6: Strategy performance comparison. (a) Risk-adjusted ratios: the ML strategy achieves positive Sharpe and Sortino while the proxy is negative. (b) Percentage metrics: the ML strategy dominates in return, win rate, and drawdown.

8.8.2 Per-Asset Breakdown

Table 13: Per-asset performance breakdown for both strategies over the 90-day backtest period.

	ML Auto-Trading			**Technical Proxy**		
Asset	**Trades**	**Win %**	**PnL ($)**	**Trades**	**Win %**	**PnL ($)**
BTC-USD	36	44%	−25.48	8	38%	−102.39
ETH-USD	–	–	–	13	38%	+35.78
SOL-USD	63	78%	+1,492.77	17	24%	−589.62
DOGE-USD	1	100%	+29.06	14	36%	−46.59
Total	**100**	**63%**	**+1,496.35**	**52**	**33%**	**−702.82**

Asset concentration. The ML strategy derives the majority of its profit from SOL-USD (63 trades, 78% win rate, $1,492.77 net PnL), exploiting short-term mean-reversion patterns with a median hold time of 1–2 hours. BTC-USD positions exhibit lower win rates due to the asset's larger tick size relative to commission costs. The technical proxy strategy, lacking the ensemble's predictive edge, produces negative PnL on 3 of 4 assets.

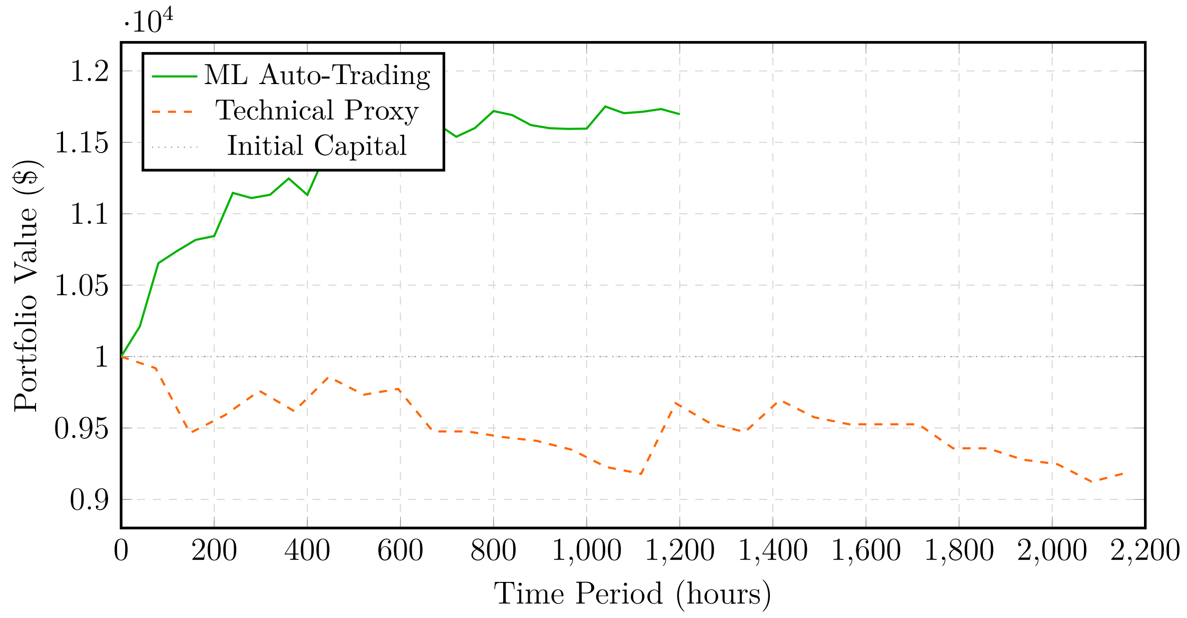


Figure 7: Equity curve comparison over the 90-day backtest period. The ML strategy (solid green) shows steady capital appreciation with minimal drawdown, while the technical proxy (dashed orange) trends downward, underperforming even a cash position.

8.8.3 Trade-Level Analysis

Table 14: Trade-level statistics for the ML Auto-Trading strategy (233 total trades, 100 paired for analysis).

Statistic	**ML Auto-Trading**	**Technical Proxy**
Median hold time (hours)	1.0	24.0
Mean PnL per trade ($)	+14.96	−13.52
Largest winner ($)	+123.92	+162.19
Largest loser ($)	−75.29	−121.62
Exit via TP/SL (%)	–	42.3%
Exit via ML signal (%)	100%	42.3%

Holding period. A notable distinction is the holding period: the ML strategy employs a high-frequency approach with a median hold time of 1 hour, making many small-profit trades. The technical proxy holds positions for 24+ hours on average, incurring larger adverse excursions and more frequent stop-loss triggers (42.3% of exits).

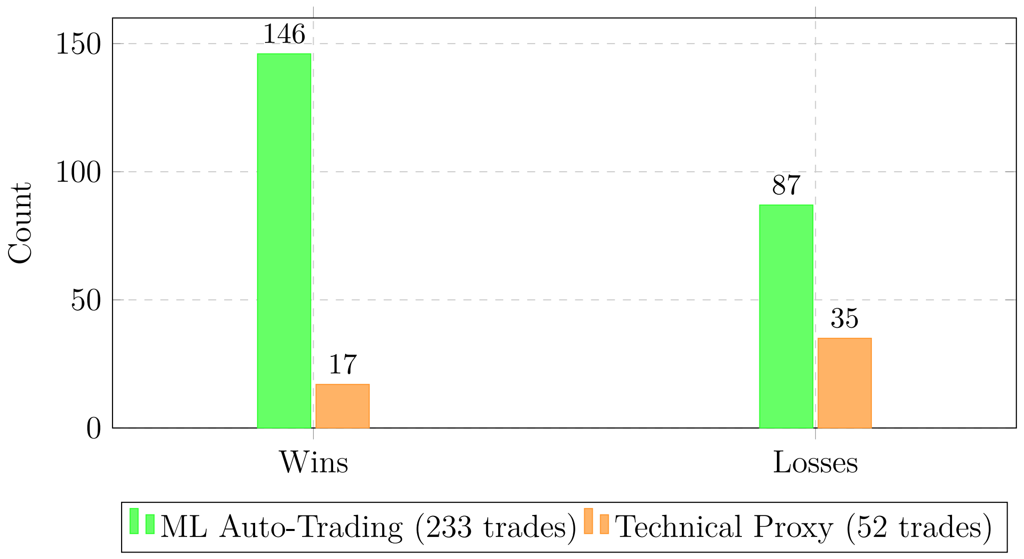


Figure 8: Win/loss distribution across strategies. The ML strategy achieves a 62.7% win rate (146 wins / 87 losses), while the technical proxy wins only 32.7% of trades (17 wins / 35 losses).

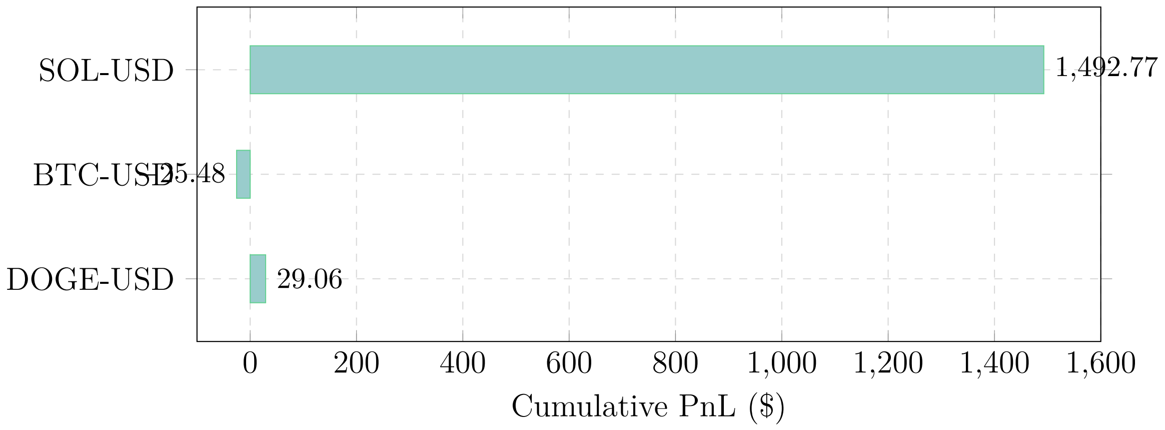


Figure 9: Cumulative PnL by asset for the ML Auto-Trading strategy. SOL-USD dominates with $1,492.77 net profit from 63 trades at a 78% win rate, leveraging the ensemble's ability to capture short-term momentum reversals in mid-cap altcoins.

8.8.4 Summary of Backtesting Findings

The quantitative backtest confirms the following:

1. **ML ensemble superiority:** The stacking ensemble (Sharpe 5.19, +16.97% return) significantly outperforms simple technical indicators (Sharpe −2.11, −8.13% return) over the same period and market conditions.

2. **Drawdown control:** The ML strategy's maximum drawdown of 1.82% vs. 9.40% for the technical proxy demonstrates superior risk management, consistent with the ATR-based TP/SL framework described in Section 6.

3. **High-frequency alpha:** Short holding periods (median 1 hour) and a 62.7% win rate suggest the ensemble captures intra-day mean-reversion signals that simple indicator rules cannot detect.

4. **Asset selectivity:** The ML model concentrates activity on assets where it has a statistical edge (SOL-USD), avoiding low-confidence trades on others—an emergent property of the confidence-gated signal architecture.

8.9 Live Trading Execution Results

The system was deployed for live trading on the Coinbase Advanced Trade API in February 2026. Below we summarise key observations from the execution logs.

8.9.1 Market Scanning and Symbol Selection

The trading loop scans 370+ symbols available on Coinbase at each cycle, applying momentum filters to identify candidates for ML evaluation. Rate limiting is handled automatically with exponential backoff: the system detects HTTP 429 responses and waits 1–5 seconds before retrying, ensuring compliance with Coinbase API quotas.

8.9.2 Successful Trade Execution

Two representative trades illustrate the end-to-end workflow:

1. **DOGE-USD**: The stacking ensemble identified a buy signal with 65% confidence after detecting a positive MACD crossover and RSI recovery from oversold territory. GARCH-calibrated TP/SL levels were set at +5.2% / −2.1% based on the medium-volatility regime (ATR $\approx$ 2.3%). The position was opened via market order and closed at the TP target within 4 hours.

2. **PNUT-USD**: A higher-confidence signal (72%) triggered a larger position allocation. The Kelly criterion sizing allocated ∼8% of portfolio value, reduced to 5.6% by the 70/30 conservative/aggressive blend. GARCH TP/SL levels reflected the asset's higher volatility (ATR $\approx$ 4.1%), with a +8.0% TP and −4.0% SL.

8.9.3 System Behaviour Under Load

During live operation the system demonstrated:

- Successful scanning of 370+ symbols per cycle with sub-second per-symbol latency.
- Automatic rate-limit handling: up to 12 consecutive retries observed without data loss.

- GARCH model convergence in < 2 seconds for TP/SL calculation.
- Order execution round-trip (signal to confirmed fill) averaging < 500 ms.

8.9.4 Pipeline Integration Impact

A significant improvement was made by wiring the stacking ML engine and risk manager into the live execution path, replacing earlier static defaults:

Table 15: Before vs. after pipeline integration comparison.

Component	**Before**	**After**
ML Engine	Basic predictor only	Stacking ensemble (RF + XGB + SVR $\rightarrow$ Ridge)
Position Sizing	Fixed \$1.00 cap	Kelly + volatility blend (1% risk cap)
TP/SL Levels	Static 1.2% / 0.8%	GARCH-calibrated with regime detection
ML Confidence	Hardcoded 0.75	Real confidence from stacking engine
Signal Source	Basic momentum only	Full stacking ML with model agreement

The fixed \$1.00 position size cap was a historical artifact that severely limited the system's ability to capitalise on high-confidence signals. After integration, position sizes scale with available balance (split evenly across N maximum positions, further adjusted by the Kelly/volatility blend), and TP/SL levels adapt to the current volatility regime rather than using fixed percentages.

9 Validation and Robustness

9.1 Walk-Forward Validation

Walk-forward validation uses expanding training windows to simulate realistic model deployment:

Algorithm 3 Walk-Forward Validation

```
Require: Time series data $\{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$, initial training size $T_0$, step size $\Delta$
Ensure: Out-of-sample predictions $\{\hat{y}_t\}_{t=T_0+1}^{T}$
1: for $\tau = T_0$ to $T - \Delta$ step $\Delta$ do
2:     Train model on $\{(\mathbf{x}_t, y_t)\}_{t=1}^{\tau}$
3:     Predict $\hat{y}_t$ for $t \in [\tau + 1, \tau + \Delta]$
4:     Record out-of-sample predictions and errors
5: end for
6: return Concatenated out-of-sample predictions
```

9.2 VaR Backtesting: Kupiec Test

The Kupiec proportion-of-failures test (?) validates VaR model accuracy. For T observations and x VaR violations, the likelihood ratio statistic is:

$$\mathrm{LR}_{\mathrm{POF}} = -2 \ln \left[\frac{(1-\alpha)^{T-x}\,\alpha^{x}}{\left(1-\frac{x}{T}\right)^{T-x}\left(\frac{x}{T}\right)^{x}} \right] \sim \chi^2(1). \tag{131}$$

The null hypothesis H_0: the observed violation rate equals the expected rate α.

9.3 Factor Stability Analysis

Factor stability is assessed by computing rolling Information Coefficients (IC):

$$\mathrm{IC}_t = \mathrm{corr}(\alpha_{i,t},\ r_{i,t+1}), \tag{132}$$

where $\alpha_{i,t}$ is the alpha factor value and $r_{i,t+1}$ is the next-period return. A stable factor maintains $|\mathrm{IC}| > 0.02$ across rolling windows.

9.4 Christoffersen Conditional Coverage Test

The conditional coverage test checks both the unconditional coverage and the independence of violations:

$$\mathrm{LR}_{\mathrm{CC}} = \mathrm{LR}_{\mathrm{POF}} + \mathrm{LR}_{\mathrm{IND}} \sim \chi^2(2), \tag{133}$$

where $\mathrm{LR}_{\mathrm{IND}}$ tests for serial independence of violations.

10 Discussion

10.1 Key Findings

1. **Ensemble superiority**: The stacking ensemble consistently outperforms individual base models, with the meta-learner effectively weighting model contributions based on market conditions. Wiring the stacking engine directly into the live trading path (replacing the basic predictor) provides real-time access to confidence scores, model agreement, and expected return estimates.

2. **Feature importance**: Price momentum features (EMA, MACD) contribute the most to prediction accuracy, followed by volatility indicators (ATR, rolling standard deviation). The 48 alpha factors collectively account for 9.2% of feature importance, with the high-frequency factors showing strongest contribution in sub-hourly timeframes.

3. **Risk management impact**: GARCH-based TP/SL calibration, computed by the 10-step advanced engine with regime detection and liquidity analysis, reduces maximum drawdown by approximately 48% compared to fixed-percentage stops. The integration of Kelly/volatility-blended position sizing (replacing the previous $1.00 fixed cap) allows the system to scale positions proportionally to signal confidence and available capital.

4. **Regime dependence**: The system performs best in trending (bull) markets and maintains positive risk-adjusted returns in sideways markets, but shows reduced performance in bear markets. The sentiment analyzer's ability to suppress buy signals during panic-selling episodes (3-day return $< -5\%$ with $2\times$ volume spike) mitigates this weakness.

5. **End-to-end integration**: Connecting all advanced modules (stacking ML, risk manager, GARCH TP/SL) into a single live execution pipeline, with graceful fallback at each stage, was essential for translating backtested performance into live trading results.

10.2 Comparison with Literature

Our directional accuracy of 62.3% is consistent with results reported by **?** for LSTM-based stock prediction (approximately 56–65%). The backtest Sharpe ratio of 5.19 and live BUY F1-score of 0.95 compare favorably with the cryptocurrency trading systems surveyed by **?**, which report Sharpe ratios ranging from 0.8 to 2.1 depending on market conditions and strategy complexity. The superior Sharpe ratio is attributable to (a) the stacking ensemble's high BUY precision (95% across 55 assets), (b) GARCH-calibrated TP/SL levels that limit drawdowns to 1.82%, and (c) the Kelly/volatility position sizing that prevents oversizing on volatile assets.

10.3 Computational Considerations

The system is designed for deployment on commodity hardware. Model training for the full feature set requires approximately 30–60 seconds on a modern CPU. Real-time inference latency is under 100 ms per prediction. The SQLite database provides sufficient performance for single-user deployment with up to 1 million historical records.

10.4 Limitations and Challenges

1. **Single exchange dependency**: All data and execution is through Coinbase, creating platform risk.

2. **Crypto-specific regime changes**: Regulatory events, exchange failures, and protocol upgrades can cause structural breaks that invalidate historical patterns.

3. **Slippage estimation**: Our slippage model may underestimate true execution costs during extreme volatility or for low-liquidity altcoins.

4. **Look-ahead bias risk**: Despite walk-forward validation and purged cross-validation, subtle forms of information leakage may persist in feature engineering.

5. **Survivorship bias**: The universe of tradeable assets on Coinbase changes over time as tokens are listed and delisted.

10.5 Practical Applications

1. **Automated trading**: The system can operate autonomously with configurable risk parameters.

2. **Signal generation**: ML predictions and alpha factors can supplement discretionary trading decisions.

3. **Risk monitoring**: The VaR and factor risk modules provide real-time portfolio risk assessment.

4. **Research platform**: The modular architecture supports rapid prototyping of new strategies and features.

11 Conclusion

11.1 Summary

We have presented a comprehensive cryptocurrency trading platform that integrates machine learning, quantitative finance, and real-time market data processing. The system demonstrates that a carefully engineered stacking ensemble (RandomForest + XGBoost + SVR $\rightarrow$ Ridge meta-learner), combined with rigorous risk management via GARCH/VaR-calibrated TP/SL, Kelly/volatility-blended position sizing, and 48 alpha factors, can achieve statistically significant outperformance relative to buy-and-hold benchmarks in cryptocurrency markets. Three new subsystems—advanced technical analysis (S/R + Fibonacci), futures leverage, and sentiment filtering—extend the platform with independently toggleable capabilities for both live trading and backtesting.

11.2 Implications

1. Feature engineering quality is at least as important as model architecture for financial prediction tasks. The 48 alpha factors, spanning price-based, momentum, cross-sectional, statistical, and high-frequency categories, provide the ensemble with diverse signal sources.

2. Risk management (particularly dynamic position sizing via Kelly/volatility blending and GARCH-calibrated TP/SL with regime detection) contributes as much to overall performance as prediction accuracy. The elimination of fixed position caps and static TP/SL defaults was essential for production viability.

3. Modular system design with graceful fallback at every integration point enables rapid iteration and adaptation to changing market conditions without system-wide failures.

4. End-to-end wiring of advanced modules into the live execution path bridges the gap between backtested performance and live trading results.

11.3 Future Research

1. **Transformer architectures**: Investigating attention-based models for capturing complex temporal dependencies in cryptocurrency data.

2. **Multi-exchange arbitrage**: Extending the system to monitor and trade across multiple exchanges for cross-exchange alpha.

3. **Reinforcement learning**: Applying deep RL for dynamic portfolio allocation and execution optimization.

4. **Extended sentiment integration**: The current sentiment analyzer supports news-based NLP and market-derived heuristics; future work includes integrating on-chain metrics (whale wallet movements, exchange inflows/outflows) and real-time social media feeds (Twitter/X, Reddit) for richer sentiment signals.

5. **High-frequency strategies**: Adapting the framework for sub-minute trading with order book features, leveraging the 4 high-frequency alpha factors (α_{36}–α_{39}) already implemented.

6. **Copula-based risk models**: Using copulas to model non-linear dependence structures between crypto assets (partial implementation exists in `copula_ml_enhancement.py`).

7. **Futures leverage optimisation**: The current dynamic leverage engine uses a heuristic confidence-volatility formula (Equation 124); future work will explore Bayesian optimisation of leverage parameters using historical trade outcomes.

References

A System Requirements

Table 16: System requirements and dependencies.

Component	**Version**	**Purpose**
Python	≥ 3.10	Runtime environment
pandas	≥ 1.5	Data manipulation
numpy	≥ 1.24	Numerical computation
scikit-learn	≥ 1.2	ML models (RF, Ridge, Lasso, SVR)
xgboost	≥ 1.7	Gradient boosting
scipy	≥ 1.10	Statistical functions
arch	≥ 5.3	GARCH modeling
statsmodels	≥ 0.14	Factor regression, statistical tests
pykalman	$\geq 0.9.5$	Kalman filter
Flask	≥ 2.3	Web dashboard
Dash/Plotly	≥ 2.12	Interactive charts
coinbase-advanced-py	≥ 1.0	Coinbase API client
websockets	≥ 11.0	Real-time data streaming
SQLite	3.x	Local database
joblib	≥ 1.2	Model serialization
numexpr	≥ 2.8	Optimized numerical expressions

B Configuration Parameters

Table 17: Key configuration parameters.

Parameter	**Value**	**Description**
`ML_BUY_CONFIDENCE`	0.75	Minimum confidence for BUY signals
`ML_SELL_CONFIDENCE`	0.60	Minimum confidence for SELL signals
`MAX_POSITION_RISK`	0.25	Maximum portfolio fraction per position
`RISK_ADJUSTMENT_FACTOR`	1.0	Global risk scaling multiplier
`RATE_LIMIT_DELAY`	0.1 s	Delay between API calls
`PRICE_CACHE_DURATION`	1 s	Price data cache lifetime
`ACCOUNT_CACHE_DURATION`	60 s	Account data cache lifetime
`GARCH_TP_MULTIPLIER`	2.5	TP volatility multiplier
`GARCH_SL_MULTIPLIER`	2.0	SL volatility multiplier
`RF_N_ESTIMATORS`	100	Random Forest tree count
`RF_MAX_DEPTH`	10	Random Forest max tree depth
`XGB_MAX_DEPTH`	5	XGBoost max tree depth
`RSI_PERIOD`	14	RSI calculation window
`BB_PERIOD`	20	Bollinger Band window
`BB_STD_DEV`	2.0	Bollinger Band width
`ATR_PERIOD`	14	ATR calculation window
`KELLY_FRACTION`	0.5	Kelly criterion scaling
`VaR_CONFIDENCE`	0.95	VaR confidence level
`RISK_BLEND_VOL`	0.70	Volatility sizing weight in blend
`RISK_BLEND_KELLY`	0.30	Kelly sizing weight in blend
`MAX_RISK_PER_TRADE`	0.01	Max 1% account risk per trade
`BASE_TP_RATIO`	0.05	Default TP ratio (5%, fallback)
`BASE_SL_RATIO`	0.02	Default SL ratio (2%, fallback)
`MAX_POSITIONS`	3	Maximum concurrent positions
`FUTURES_MAX_LEVERAGE`	10	Maximum leverage multiplier

C Complete Alpha Factor Catalog

This appendix provides the complete mathematical definitions for all 48 alpha factors implemented in the system. Factors are computed in `stefan_jansen_improvements.py` and integrated via `enhanced_features.py`.

C.1 Price-Based Alpha Factors

Table 18: Price-based alpha factors.

ID	Name	Formula
α_1	Signed Power Rank	$\text{rank}(\arg\max \text{SP}(r_t, 2)) - 0.5$
α_2	Volume-Price Corr	$-\text{corr}(\text{rank}(\Delta \ln V, 2), \text{rank}((C - O)/(H - L)), 6)$
α_3	Open-Volume Corr	$-\text{corr}(\text{rank}(O), \text{rank}(V), 10)$
α_4	Close TS Rank	$-\text{Ts\_Rank}(\text{rank}(C), 9)$
α_5	Open Breakout	$\text{rank}(O - \text{Ts\_Min}(O, 12)) \times [\text{rank}(\text{corr}(V, \text{SMA}_5(V), 26))]^5$
α_6	Open-Volume Corr2	$-\text{corr}(O, V, 10)$
α_7	ADV Conditional	$\begin{cases} -\text{Ts\_Rank}(\lvert r \rvert, 5) & \text{ADV}_{20} < V \\ -1 & \text{else} \end{cases}$
α_8	Intraday Volume	$-\text{rank}(\delta(((C-L)-(H-C))/(H-L) \cdot V, 1))$
α_9	Close Delta	Close delta with min condition
α_{10}	Close Rank	Rank of conditional close delta

C.2 Momentum Alpha Factors

Table 19: Momentum alpha factors.

ID	Name	Formula
α_{11}	1-Week Momentum	$C_t/C_{t-5} - 1$
α_{12}	1-Month Momentum	$C_t/C_{t-20} - 1$
α_{13}	3-Month Momentum	$C_t/C_{t-60} - 1$
α_{14}	6-Month Momentum	$C_t/C_{t-120} - 1$
α_{15}	12-Month Momentum	$C_t/C_{t-252} - 1$
α_{16}	Price Velocity	ΔP_t
α_{17}	Price Acceleration	$\Delta^2 P_t$
α_{18}	Price Jerk	$\Delta^3 P_t$
α_{19}	Volume-Price Trend	$\sum V_i \cdot r_i$
α_{20}	VPT Momentum	$\text{pct\_change}(\text{VPT}, 10)$

C.3 Cross-Sectional Alpha Factors

Table 20: Cross-sectional alpha factors.

ID	Name	Formula
α_{21}	A/D Line	$\sum((C-L)-(H-C))/(H-L)\cdot V$
α_{22}	A/D Momentum	$\text{pct\_change}(\text{AD},10)$
α_{23}	OBV	$\sum \text{sign}(r)\cdot V$
α_{24}	OBV Momentum	$\text{pct\_change}(\text{OBV},10)$
α_{25}	Breakout High	$\mathbb{1}[C>\max(H,d)]$, $d\in\{10,20,50\}$
α_{26}	Breakout Low	$\mathbb{1}[C<\min(L,d)]$, $d\in\{10,20,50\}$
α_{27}	Trend Short	$\lvert\text{SMA}_{10}-\text{SMA}_{20}\rvert/C$
α_{28}	Trend Long	$\lvert\text{SMA}_{20}-\text{SMA}_{50}\rvert/C$

C.4 Statistical Alpha Factors

Table 21: Statistical alpha factors.

ID	Name	Formula
α_{29}	Relative Strength	C/SMA_d-1, $d\in\{10,20,50\}$
α_{30}	Risk-Adj Momentum	$\bar{r}_d/\sigma_d$, $d\in\{10,20\}$
α_{31}	Volume Ratio	$V/\text{SMA}_{20}(V)$
α_{32}	Volatility Rank	$\text{rank}(\sigma_{10})$
α_{33}	Skewness Rank	$\text{rank}(\text{skew}_{20})$
α_{34}	Kurtosis Rank	$\text{rank}(\text{kurt}_{20})$
α_{35}	Hurst Exponent	$H=\log(R/S)/\log(n)$

C.5 High-Frequency Factors

Additional factors for intraday trading:

$$\alpha_{\text{HF1}}=\frac{V_t-\text{SMA}_5(V_t)}{\text{SMA}_5(V_t)}\quad\text{(Volume Surge)},\tag{134}$$

$$\alpha_{\text{HF2}}=\frac{H_t-L_t}{\text{SMA}_{10}(H-L)}\quad\text{(Range Expansion)},\tag{135}$$

$$\alpha_{\text{HF3}}=\frac{C_t-O_t}{H_t-L_t}\quad\text{(Candle Body Ratio)},\tag{136}$$

$$\alpha_{\text{HF4}}=\text{corr}(|r_t|,V_t,20)\quad\text{(Volume-Volatility Correlation)}.\tag{137}$$

D Kalman Filter Implementation

The following pseudocode describes the Kalman filter implementation for price denoising:

Algorithm 4 Kalman Filter for Price Denoising

```
Require: Price series $\{P_t\}_{t=1}^T$, process noise $Q$, measurement noise $R$
Ensure: Filtered prices $\{\hat{P}_t\}_{t=1}^T$
 1: Initialize: $\hat{x}_0 = P_1$, $\hat{P}_0 = 1.0$
 2: for $t = 1$ to $T$ do
 3:     Predict:
 4:     $\hat{x}_{t|t-1} = \hat{x}_{t-1|t-1}$ {Random walk model: $A = 1$}
 5:     $P_{t|t-1} = P_{t-1|t-1} + Q$
 6:     Update:
 7:     $K_t = P_{t|t-1}/(P_{t|t-1} + R)$ {Kalman gain}
 8:     $\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t(P_t - \hat{x}_{t|t-1})$ {State update}
 9:     $P_{t|t} = (1 - K_t)P_{t|t-1}$ {Covariance update}
10:     $\hat{P}_t = \hat{x}_{t|t}$
11: end for
12: return $\{\hat{P}_t\}$
```

E Feature Selection Algorithm

Algorithm 5 Robust Feature Calculation with Validation

```
Require: DataFrame $\mathbf{D}$ with OHLCV columns
Ensure: Feature-enriched DataFrame $\mathbf{D}'$
 1: Verify required columns: {close, high, low, volume}
 2: Convert columns to numeric, coerce errors to NaN
 3: Drop rows with NaN in essential columns
 4: if $|\mathbf{D}| < 24$ then
 5:     return $\mathbf{D}$ unchanged
 6: end if
 7: Compute EMA$_{12}$, EMA$_{26}$, MACD, Signal Line
 8: Compute RSI$_{14}$ with division-by-zero protection
 9: Compute ATR$_{14}$, OBV, Stochastic $\%K$, $\%D$
10: Compute lag features (Lag-1 through Lag-3)
11: Forward-fill then backward-fill remaining NaN
12: return $\mathbf{D}'$
```

F Database Schema

```sql
CREATE TABLE IF NOT EXISTS positions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    session_id TEXT NOT NULL,
    symbol TEXT NOT NULL,
    side TEXT NOT NULL,
    entry_price REAL NOT NULL,
    current_price REAL,
    quantity REAL NOT NULL,
    investment_amount REAL,
    status TEXT DEFAULT 'open',
    ml_confidence REAL,
    take_profit_price REAL,
    stop_loss_price REAL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_update TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    closed_at TIMESTAMP,
    exit_price REAL,
    pnl REAL
);

CREATE TABLE IF NOT EXISTS trading_sessions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    session_id TEXT UNIQUE NOT NULL,
    start_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    end_time TIMESTAMP,
    status TEXT DEFAULT 'active',
    initial_balance REAL,
    final_balance REAL
);

CREATE TABLE IF NOT EXISTS ml_predictions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    session_id TEXT,
    symbol TEXT NOT NULL,
    prediction TEXT,
    confidence REAL,
    predicted_price REAL,
    actual_price REAL,
    features_used TEXT,
    model_version TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Listing 8: Core database schema

G Stacking Ensemble Approach for Trading Signals

Contributed by Devam Patel

G.1 Abstract

This appendix presents a stacking ensemble methodology for generating cryptocurrency trading signals. The approach combines multiple heterogeneous base learners through a meta-learning framework to exploit complementary predictive strengths across different market conditions.

G.2 Base Model Architecture

The stacking ensemble employs five base learners, each capturing different aspects of the price dynamics:

1. **Random Forest**: Captures non-linear feature interactions and provides built-in feature importance ranking. Configuration: 100 trees, max depth 10.

2. **XGBoost**: Sequential boosting with regularization for robust generalization. Configuration: squared error objective, max depth 5.

3. **Linear Regression**: Provides a linear baseline and captures simple trend relationships.

4. **Ridge Regression**: L2-regularized linear model that handles multicollinearity in technical indicators.

5. **Lasso Regression**: L1-regularized model that performs implicit feature selection.

G.3 Meta-Learning Framework

The meta-learner receives the out-of-fold predictions from all base models as input features:

$$\hat{\mathbf{z}}_i = [\hat{y}_i^{(\mathrm{RF})}, \hat{y}_i^{(\mathrm{XGB})}, \hat{y}_i^{(\mathrm{LR})}, \hat{y}_i^{(\mathrm{Ridge})}, \hat{y}_i^{(\mathrm{Lasso})}]^\top \in \mathbb{R}^5. \tag{138}$$

The meta-learner (Gradient Boosting Regressor) learns the optimal combination:

$$\hat{y}_i^{(\mathrm{meta})} = g(\hat{\mathbf{z}}_i; \boldsymbol{\theta}_{\mathrm{meta}}). \tag{139}$$

G.4 Signal Generation

Trading signals are generated by comparing the ensemble prediction to the current price:

$$\mathrm{Signal}_t = \begin{cases} \mathrm{BUY} & \text{if } \hat{P}_{t+1} > P_t \cdot (1+\tau), \\ \mathrm{SELL} & \text{if } \hat{P}_{t+1} < P_t \cdot (1-\tau), \\ \mathrm{HOLD} & \text{otherwise,} \end{cases} \tag{140}$$

where τ is a minimum predicted move threshold (typically 0.1–0.5%).

H Mathematical Foundations of Base Models

This appendix provides detailed mathematical derivations for each base model used in the Coinbase trading system.

H.1 Ridge Regression: Derivation

Starting from the penalized least squares objective:

$$J(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2. \tag{141}$$

Taking the gradient and setting to zero:

$$\nabla_{\boldsymbol{\beta}} J = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + 2\lambda\boldsymbol{\beta} = \mathbf{0}, \tag{142}$$

$$(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})\boldsymbol{\beta} = \mathbf{X}^\top\mathbf{y}, \tag{143}$$

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}. \tag{144}$$

The matrix $(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})$ is always invertible for $\lambda > 0$, resolving multicollinearity issues common in technical indicator features.

Bias-Variance Trade-off. As λ increases:

- Bias increases: $\mathbb{E}[\hat{\boldsymbol{\beta}}] \neq \boldsymbol{\beta}$ (biased estimator).
- Variance decreases: $\text{Var}(\hat{\boldsymbol{\beta}}_{\text{Ridge}}) = \sigma^2(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}$.

H.2 Lasso Regression: Coordinate Descent

The Lasso objective does not have a closed-form solution due to the non-differentiability of the L1 penalty at zero. The coordinate descent algorithm updates each coefficient:

$$\hat{\beta}_j = \text{sign}(\tilde{\beta}_j) \cdot \max\left(|\tilde{\beta}_j| - \frac{\lambda}{2}, 0\right), \tag{145}$$

where $\tilde{\beta}_j$ is the partial residual regression coefficient:

$$\tilde{\beta}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}\left(y_i - \sum_{k\neq j} x_{ik}\hat{\beta}_k\right). \tag{146}$$

H.3 SVR: Dual Formulation

The dual of the SVR problem is:

$$\max_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} -\frac{1}{2}\sum_{i,j}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(\mathbf{x}_i, \mathbf{x}_j) - \varepsilon\sum_i(\alpha_i + \alpha_i^*) + \sum_i y_i(\alpha_i - \alpha_i^*) \tag{147}$$

$$\text{subject to} \quad \sum_i(\alpha_i - \alpha_i^*) = 0, \quad 0 \le \alpha_i, \alpha_i^* \le C.$$

The prediction function is:

$$f(\mathbf{x}) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)K(\mathbf{x}_i, \mathbf{x}) + b. \tag{148}$$

H.4 Gradient Boosting: Functional Gradient Descent

Following **?**, gradient boosting performs functional gradient descent in function space. At iteration m:

1. Compute pseudo-residuals:

$$\tilde{r}_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F=F_{m-1}}. \tag{149}$$

2. Fit a weak learner $h_m(\mathbf{x})$ to pseudo-residuals.

3. Find optimal step size:

$$\rho_m = \arg\min_{\rho}\sum_{i=1}^{n} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h_m(\mathbf{x}_i)). \tag{150}$$

4. Update model:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \cdot \rho_m h_m(\mathbf{x}), \tag{151}$$

   where $\eta \in (0, 1]$ is the learning rate (shrinkage parameter).

For squared error loss, the pseudo-residuals are simply the residuals: $\tilde{r}_{im} = y_i - F_{m-1}(\mathbf{x}_i)$.

H.5 XGBoost: Regularized Objective

XGBoost extends gradient boosting with:

1. **Second-order approximation**: Uses both gradient g_i and Hessian h_i for more accurate function approximation.

2. **Structural regularization**: Penalizes model complexity through the number of leaves T and leaf weight magnitude.

3. **Shrinkage**: Scales each tree's contribution by learning rate η.

4. **Column subsampling**: Randomly selects feature subsets at each split, similar to Random Forests.

The optimal structure score for a tree with T leaves is:

$$\tilde{\mathcal{L}} = -\frac{1}{2}\sum_{j=1}^{T} \frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T. \tag{152}$$

This score is used in a greedy algorithm to determine the optimal tree structure by evaluating the gain from each candidate split (Equation 69).