NICOLAS PALOMARES

ML-Enhanced Crypto Trading System

Backtesting & Live Simulation Framework

A modular algorithmic trading system combining technical analysis with machine learning signal filtering. Validated through backtesting methodologies with comprehensive risk management

Table of Contents

Introduction

Algorithmic trading is a field where quantitative analysis, software engineering, and risk management intersect. Rather than focusing on predicting future prices directly, most robust trading systems are designed to exploit repeatable market behaviors while strictly controlling downside risk. This project was developed with that philosophy in mind.

The objective of this project is to design and implement a complete algorithmic trading system, starting from historical data analysis and ending with a replay-based execution framework that simulates live trading behavior. The system is not intended to place real trades automatically, but to generate structured BUY/HOLD recommendations while managing positions through predefined risk rules such as stop-loss and take-profit levels.

A key design decision of this project is the separation between **signal generation** and **trade execution**. The trading strategy itself is based on deterministic technical conditions (trend regime, momentum pullbacks, volatility and volume filters), while trade management is handled by a dedicated engine that simulates position sizing, fees, equity evolution and exit logic. This separation makes the system easier to extend and debug.

The project also addresses the limitations of purely rule-based strategies. To address this, a machine learning model is introduced as a **trade quality filter**. Instead of attempting to predict price movements, the model learns to estimate whether a given strategy signal is likely to result in a profitable trade, based on historical outcomes.

From a software perspective, the project emphasizes reproducibility and modularity. The same strategy logic is reused across backtesting, walk-forward analysis, robustness testing, and live replay execution. All components are designed to run locally from the command line, allowing users to clone the repository and reproduce results without external dependencies or persistent infrastructure.

Overall, this project serves as an end-to-end demonstration of how a quantitative trading idea can be translated into a structured, testable, and extensible trading system, with a strong focus on risk control, validation and practical execution constraints.

Development

The project was developed iteratively, with an emphasis on **systematic validation**, **risk control**, and **separation of concerns** between data handling, signal generation, execution logic and evaluation. The goal was not to optimize a single backtest result, but to progressively build a trading system that remains coherent under different market conditions and evaluation regimes.

1. D*ata Pipeline and Feature Engineering*

The system operates on historical OHLCV candlesticks retrieved from the Binance API. To overcome API limitations and ensure data completeness, a paginated data loader was implemented, allowing the collection of a full year of hourly data (January-December 2024). Raw data is validated and stored before any transformation. A dedicated feature engineering stage computes a set of technical indicators commonly used in trading, including trend indicators (EMA, SMA), momentum indicators (RSI), volatility measures (ATR, logarithmic returns) and volume-based features (normalized volume and ratios)

2. R*ule-Based Strategy Design*

The initial strategy is fully rule-based and long-only. Entry signals are generated from a combination of trend confirmation and momentum conditions, while exits are handled exclusively through **adaptive stop-loss and take-profit levels** based on ATR. Key design decisions include:

- One position at a time (no pyramiding)
- No discretionary exits: trades are closed only by stop-loss or take-profit

- Dynamic risk management via fixed fractional risk per trade
- Position sizing derived from stop distance rather than notional exposure

This design ensures that performance is driven by **trade quality and risk asymmetry**.

3. *Backtesting Framework*

A custom backtesting engine was implemented to simulate realistic trade execution. The engine tracks equity over time, applies trading fees, manages open positions and records each completed trade with full metadata (entry, exit, duration, PnL, exit reason). Performance evaluation goes beyond aggregate returns and includes:

- Maximum drawdown
- Win rate
- Trade-level statistics
- Expectancy and profit factor
- Out-of-sample (70/30 split)
- Walk-forward analysis
- Robustness testing across multiple SL/TP configurations

4. *Machine Learning as a Trade Quality Filter*

After establishing a stable rule-based baseline, a machine learning model was introduced as a **trade quality filter**. This model is trained on historical trades produced by the unfiltered strategy. For each trade entry, features are extracted from the market state at entry time, and the target variable is derived from realized trade outcomes using a profitability-based labeling scheme. Key characteristics of this approach:

- The model never creates trades on its own
- It acts as a probabilistic filter on rule-based BUY signals
- Strategy logic, risk management, and exit rules remain unchanged
- This design preserves deterministic core behavior while adding statistical validation

The system remains fundamentally rule-driven, with ML acting as a statistical confirmation layer.

5. *Live Simulation Execution Architecture*

The live component mirrors the backtest architecture as closely as possible. A streaming data feed processes closed candles sequentially, updates features in real time, evaluates signals, applies the ML filter, and forwards approved signals to the trade engine. By maintaining parity between backtest and live logic, the system minimizes simulation-to-production discrepancies and allows direct comparison between historical and live behavior.

Results

The strategy was evaluated using one full year of hourly market data for **ETHUSDT**, covering the period from **January 1st** to **December 31st, 2024**. All tests incorporate realistic assumptions, including trading fees, fixed fractional risk and volatility-adjusted exists. Configuration:

- Instrument: ETHUSDT
- Timeframe: 1 hour
- Initial capital: $10,000.00
- Risk per trade: 1.0%
- Trading fee: 0.10%
- Stop-loss: 1.8 x ATR
- Take-profit: 2.4 x ATR
- Data source: Historical OHLCV candles
- Total observations: 8,761 hourly bars
- Coverage: Entire 2024 calendar year

This configuration was selected based on robustness testing rather than pure return maximization. The full-period backtest demonstrates strong performance with controlled downside risk:

- Total return: +70.08%
- Maximum drawdown: -2.97%
- Win rate: 75.71%
- Initial equity: $10,000.00
- Final equity: $17,008.08
- Net P&L: +$7,008.08

A total of **70 trades** were executed over the year, resulting in moderate trade frequency and multi-day holding periods:

- Trades executed: 70
- Trade win rate: 72.86%

- Average trade return: +0.56%
- Average trade duration: ~3.5 days
- Expectancy per trade: +0.56%
- Average win: +2.83%
- Average loss: -5.51%
- Profit factor: 1.38

Despite larger average losses per losing trade, the high win rate (72.86%) combined with disciplined risk sizing maintains positive expectancy and a profit factor of 1.38. To evaluate generalization, the strategy was tested on a strictly forward out-of-sample segment. **OOS performance (70/30 split)**:

- Total return: +6.36%
- Maximum drawdown: -2.97%
- Win rate: 54.17%
- Trades executed: 24
- Trade win rate: 70.83%
- Average trade return: +1.22%
- Expectancy: +1.22%
- Profit factor: 1.99

A **walk-forward evaluation using 9 rolling windows** was performed to assess stability across different market regimes. Out-of-Sample statistics:

- Mean total return: ~15.0%
- Mean OOS return per window: ~3.2%
- Mean maximum drawdown: -1.11%
- Mean win rate: 73.21%

In the **Robustness test**, the strategy was stress-tested across multiple stop-loss and take-profit combinations. These are the key observations:

- Robust performance persists across a wide parameter range
- No single configuration dominates results
- The selected (1.8/2.4 ATR) setup offers a strong balance between return, drawdown, and win rate

Conclusions

This project presents the end-to-end development of a systematic trading framework designed to generate structured, risk-controlled trading decisions on hourly cryptocurrency data. Using a full year of ETHUSDT market data from 2024, the strategy combines deterministic rule-based signals with volatility-adjusted risk management and a machine learning trade-quality filter.

The core system applies ATR-based stop-loss and take-profit levels, fixed fractional risk per trade, and realistic fee assumptions, resulting in a +70.08% total return with a maximum drawdown of -2.97% over the full backtest period. Trade-level analysis shows stable behavior across 70 executed trades, with a 72.86% win rate, positive expectancy, and a profit factor of 1.38.

Out-of-sample validation and walk-forward testing confirm that performance is not driven by a single optimization window, as the system preserves positive returns, controlled drawdowns, and consistent trade quality across rolling segments. Robustness tests across multiple stop-loss and take-profit configurations further demonstrate that profitability is not dependent on narrowly tuned parameters.

The separation between signal generation and trade execution proved to be a valuable architectural decision, enabling independent testing and modification of strategy rules without affecting risk management logic. The machine learning component successfully functions as a statistical filter rather than a predictive engine, maintaining the interpretability and transparency of the rule-based core while improving signal quality.

The emphasis on modularity and reproducibility allows the system to be deployed and validated independently by other users. The configuration-driven design through config.yaml enables rapid experimentation with different parameters without code modification, while the unified codebase shared between backtesting and live simulation minimizes the risk of simulation-to-production discrepancies.

This project demonstrates a disciplined approach to quantitative trading system design, prioritizing risk-adjusted performance, validation rigor, architectural clarity, and controlled complexity over superficial return maximization. The results suggest that combining deterministic strategy rules with machine learning filters can yield robust performance when validated through comprehensive out-of-sample and walk-forward testing methodologies.

Limitations & Future Improvements

Despite the encouraging results obtained in this project, several important limitations must be acknowledged, and multiple directions for future development have been identified.

Data and Market Coverage

The evaluation is restricted to a single calendar year (2024) and primarily one trading instrument (ETHUSDT), leaving multi-cycle and cross-asset robustness untested. A full market cycle typically spans multiple years and includes bull markets, bear markets, and consolidation periods. Testing the strategy across 2020-2024 would provide stronger evidence of adaptability to different market regimes. Additionally, validating performance across other cryptocurrency pairs (BTCUSDT, BNBUSDT) and potentially traditional asset classes would confirm whether the strategy logic generalizes beyond a single instrument.

Execution Modeling

While trading fees are included in all simulations, execution assumptions remain simplified and do not account for slippage, liquidity constraints, or spread variability. In live trading, especially during periods of high volatility or low liquidity, actual fill prices may deviate significantly from the assumed entry and exit levels. Future iterations should incorporate more realistic execution models, including order book dynamics, market impact modeling, and variable spreads that widen during volatile conditions.

Risk Management Scope

Risk management currently operates at the individual trade level rather than at the portfolio level. A more sophisticated approach would include maximum drawdown controls, daily loss limits, and position correlation analysis when trading multiple instruments simultaneously. Additionally, the current system uses static stop-loss and take-profit multipliers, whereas adaptive mechanisms that adjust based on recent volatility or market regime could improve risk-adjusted returns.

Machine Learning Component

The machine learning filter, although implemented as a trade quality classifier rather than a signal generator, is still trained on historical trades from the same base strategy. This introduces potential regime dependency risk, as the model may not generalize well to market conditions that differ substantially from the training period. Future work should explore incremental learning approaches that periodically retrain the model on recent data, as well as regime detection mechanisms that adjust model parameters or thresholds based on detected market states.

Strategy Parameter Adaptation

Currently, all strategy parameters remain static throughout the backtest and live simulation periods. An adaptive approach could dynamically adjust ATR multipliers, RSI thresholds, or moving average periods based on recent market volatility, trend strength, or regime classification. This would allow the strategy to remain responsive to changing market conditions without manual intervention.

Based on the identified limitations, the following improvements are recommended:

1. Multi-year dataset expansion
2. Cross-asset validation
3. Enhanced execution modeling
4. Portfolio-level risk management
5. Regime detection integration
6. Adaptive parameter optimization
7. Structured model retraining

Despite these limitations, the current implementation provides a solid foundation for further research and development. The modular architecture and comprehensive validation framework ensure that future enhancements can be integrated and tested systematically without compromising the integrity of the existing codebase.

References

- Python Software Foundation. (2024). Python Programming Language (Version 3.13.11). https://www.python.org/
- McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference, 56-61. https://pandas.pydata.org/
- Harris, C. R., Millman, K. J., van der Walt, S. J., et al. (2020). Array programming with NumPy. Nature, 585(7825), 357-362. https://numpy.org/
- Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830. https://scikit-learn.org/
- Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794. https://xgboost.readthedocs.io/
- Binance. (2024). Binance API Documentation. https://binance-docs.github.io/apidocs/