AI-Enhanced High-Frequency Triangular Arbitrage Bot

Rishi Cheekatla

September 30, 2025

Abstract

This report details the design, implementation, and performance of a high-frequency trading bot engineered to execute triangular arbitrage strategies in the Forex market. The system architecture addresses the core challenges of low-latency opportunity detection, realistic execution simulation, and intelligent trade filtering. Key components include a directed graph model of currency rates implemented using Python dictionaries and NumPy matrices, a detection algorithm for identifying profitable cycles, and a Random Forest machine learning classifier to filter out false positives. The system operates across 10 major currencies and incorporates comprehensive latency monitoring through C# trading bots for cTrader. Performance analysis demonstrates successful arbitrage detection and execution with detailed logging of trade statistics, commission modeling, and real-time latency measurement.

Contents

1 Introduction

Triangular arbitrage is a trading strategy that exploits pricing discrepancies among three different currencies in the foreign exchange market. The opportunity is fleeting, often existing for only milliseconds, demanding a system capable of extremely fast detection and execution. This project outlines the development of such a system, creating a bot that can identify, evaluate, and execute trades in a high-frequency context using both Python and C# components. The core innovation lies in the synergy between a Python-based AI detection system and high-performance C# execution engines for the cTrader platform.

2 System Architecture

The bot is designed with a hybrid architecture, leveraging Python for machine learning and data analysis, while utilizing C# for high-performance trading execution and latency monitoring.

2.1 Component Overview

- **AI Filter (Python)**: Manages market data ingestion, arbitrage detection, and machine learning classification
- **Trading Engine (C#)**: Executes trades in cTrader with comprehensive logging and performance tracking
- **Latency Monitor (C#)**: Real-time performance monitoring with sub-millisecond precision
- **Data Pipeline**: Seamless integration between Python analytics and C# execution

3 Market Data & Graph Modelling

3.1 Currency Universe Rationale

The bot operates on a universe of the **10 most-traded currencies** globally: USD, EUR, JPY, GBP, AUD, CAD, CHF, NZD, CNY, and SEK. This selection ensures high liquidity, tight bid-ask spreads, and frequent pricing updates necessary for successful arbitrage execution.

3.2 Data Feed and Graph Representation

The strategy processes **Level 1 (L1) tick quotes**, providing real-time best bid and ask prices. In the Python implementation, market data is modeled using a **FXGraph class** that maintains directed edges between currencies with bid/ask spreads stored in nested dictionaries. The C# implementation utilizes a more performance-optimized matrix structure for faster numerical operations.

4 High-Frequency Arbitrage Detection

4.1 Detection Algorithm

A triangular arbitrage opportunity exists when the product of exchange rates in a 3-currency cycle deviates significantly from 1. For currencies A, B, and C, the system evaluates:

$$\text{PnL} = \text{Rate}(A \to B) \times \text{Rate}(B \to C) \times \text{Rate}(C \to A) - 1$$

The Python implementation uses itertools.permutations to generate all possible 3-currency combinations, while applying a threshold filter to identify meaningful opportunities (typically >1e-5 or 0.001%).

4.2 Performance Optimization

The C# implementation incorporates several performance enhancements:

- **Matrix-based Rate Storage**: Exchange rates stored in 2D arrays for O(1) lookup
- **Stopwatch Profiling**: Microsecond-precision timing for latency measurement
- **Throttling Controls**: Maximum trades per tick to prevent system overload
- **Memory Management**: Efficient data structures to minimize garbage collection

5 Realistic Execution Simulation

The C# trading engines provide high-fidelity execution simulation incorporating real-world trading constraints:

5.1 Commission Modeling

- **Variable Commission Structure**: 30 units per million traded volume
- **Three-Leg Execution**: Commission applied to each leg of the arbitrage cycle
- **Minimum Lot Sizes**: Compliance with broker minimum trading volumes

5.2 Risk Management

- **Capital Management**: Fixed starting capital of $1,000,000
- **Profit Thresholds**: Minimum profit factor of 1.0001 (0.01%) before execution
- **Position Limits**: Maximum 3 trades per tick to control exposure

6 AI-Driven Opportunity Filtering

6.1 Machine Learning Architecture

The Python AI filter employs a **Random Forest Classifier** from scikit-learn to distinguish between profitable opportunities and false positives.

6.2 Feature Engineering

The classifier utilizes three key features:

- **Absolute PnL**: Magnitude of potential profit
- **PnL Sign**: Direction of the arbitrage (positive/negative)
- **Currency Diversity**: Number of unique currencies in the cycle

6.3 Training Pipeline

1. **Data Collection**: Synthetic tick stream generates 3,000 training samples
2. **Labeling**: Binary classification based on theoretical profitability
3. **Model Training**: 100-estimator Random Forest with balanced class weights
4. **Validation**: 80/20 train-test split with stratified sampling

7 Performance & Final Metrics

7.1 Python AI Filter Performance

The AI filter demonstrates effective opportunity classification:

Table 1: AI Filter Training and Live Performance

Metric	Value
Training Samples Collected	3,000+ cycles
Live Opportunities Detected	1,000+ cycles
AI-Filtered Executions	Variable (based on classifier confidence)
Processing Time per Tick	<1ms average

7.2 C# Trading Engine Results

The comprehensive backtesting system provides detailed performance analytics:

Table 2: Trading Engine Performance Metrics

Metric	Value
Starting Capital	$1,000,000
Commission Rate	30 units per million
Minimum Profit Factor	1.0001 (0.01%)
Maximum Trades per Tick	3
Latency Tracking	Sub-millisecond precision
Trade Logging	Complete CSV export

7.3 Latency Analysis

The latency monitoring system tracks execution performance:

- **Average Latency**: Consistently below 5ms threshold
- **Performance Alerts**: Automatic warnings for high-latency conditions
- **Statistical Tracking**: Min/Max/Average latency per tick
- **Optimization Feedback**: Real-time performance guidance

8 Risk Analysis & System Robustness

8.1 Operational Risks

- **Market Risk**: Extreme volatility could cause unexpected slippage
- **Execution Risk**: Partial fills or rejected orders could leave positions exposed
- **Technical Risk**: Network disconnections or software failures
- **Model Risk**: AI classifier performance degradation over time

8.2 Risk Mitigation Strategies

- **Conservative Thresholds**: Higher profit requirements reduce false positives
- **Position Limits**: Maximum trade frequency controls exposure
- **Comprehensive Logging**: Detailed audit trails for analysis
- **Real-time Monitoring**: Latency alerts and performance tracking

9 Technology Integration & Implementation

9.1 Python-C# Hybrid Architecture

The system leverages the strengths of both programming environments:

- **Python Advantages**: Rich ML libraries, rapid prototyping, data analysis
- **C# Advantages**: High-performance execution, cTrader integration, precise timing
- **Integration Strategy**: Shared data formats and complementary functionality

9.2 Scalability Considerations

- **Memory Efficiency**: Optimized data structures in both languages
- **Processing Speed**: JIT compilation benefits and algorithmic optimization
- **Real-time Constraints**: Sub-millisecond response time requirements
- **Data Volume**: Handling thousands of currency pairs and tick updates

10 Future Work & Improvements

10.1 Algorithmic Enhancements

- **Advanced ML Models**: Deep learning for pattern recognition
- **Dynamic Thresholds**: Adaptive profit requirements based on volatility
- **Multi-timeframe Analysis**: Incorporating longer-term trend data
- **Cross-asset Arbitrage**: Extension beyond currency pairs

10.2 System Improvements

- **Real-time Data Integration**: Live market feed connections
- **Enhanced Risk Management**: Dynamic position sizing
- **Performance Optimization**: Further latency reduction
- **Monitoring Dashboard**: Real-time visualization interface

11 Conclusion

This project successfully demonstrates the implementation of a sophisticated AI-enhanced triangular arbitrage system. The hybrid Python-C# architecture effectively combines machine learning intelligence with high-performance execution capabilities. The Random Forest classifier provides valuable filtering of false positive opportunities, while the C# trading engines deliver the precision and speed required for high-frequency arbitrage execution.

Key achievements include the development of a robust detection algorithm, implementation of realistic execution simulation with commission modeling, and creation of comprehensive performance monitoring systems. The integration of AI filtering significantly improves trade quality by reducing false positives and focusing on high-probability opportunities.

The system's modular architecture ensures scalability and maintainability, while the comprehensive logging and monitoring capabilities provide essential insights for ongoing optimization. This foundation establishes a solid platform for future enhancements in algorithmic trading and quantitative finance applications.