Low Latency Cryptocurrency Trading System with Predictive Models

Ken Chen kc3037@nyu.edu
Nicole Zhao xz2616@nyu.edu
Yuqin Wang yw3875@nyu.edu
Linxia Li ll4764@nyu.edu
Xinyu Guo xg693@nyu.edu

Abstract

We introduce a real-time, low-latency python-based trading system for cryptocurrency arbitrage strategies. The main challenge is to efficiently download historical data, record live data, train predictive model, and run model in real-time. In the following sections of paper, we would discuss about processes in details and focus on optimizing run time. Our code is available at https://github.com/ck2w/cryptopy.

1. Introduction

Low latency is the use of algorithmic trading to react to market events faster than the competition to increase profitability of trades. If a trader has a lot of time and resources, the best solution for speed will be coded in FPGA or at least using C++. But if we aim to engage with 20+ crypto exchanges, to go to market quickly, and to implement continuous strategy and performance tuning, Python would be a better choice.

Currently, there are more than 10 liquid crypto exchanges (Binance, Coinbase, FTX, Huobi, OKX etc.), one trading system might need to connect to many different markets and adapt to different APIs. Besides, there are hundreds of trading instruments (BTC/BCH/ETH/EOS) and types of instruments (spot/futures/swaps). Thus, arbitrage strategies are more profitable than traditional market. Nearly all exchanges connections are based on REST and WebSocket API. The latency from sending order to getting exchange response are at the level of 10-30ms. It implies that a python-based trading system could satisfy the latency needs.

Based on VeighNa open source quantitative trading sytem framework [1], we build a crypto trading system that could be used to trade arbitrage trading strategies using Python. The system has connected to OKX.com crypto exchange based on REST/WebSocket API. We build functions including data recording, signal, and position calculation. Historical minute bar data and tick order book data are used to train predictive models to predict price changes and are made for trading strategies in live trading. We control the internal latency to the level of 10-20ms (from receiving data to sending orders) with complicated models in our strategy.

2. Dataset

We rely on multiple datasets including publicly available historical data and recorded tick data we collect to test the predictive model, backtest the strategy and facilitate the live trading. Since we will test our trading on OKX.com, we only collect data from the website's API.

Minute bar data. Each 1-minute bar has OHLCV (open/high/low/close/volume) data which is aggregated from trades executed on each trading instruments. With API provided by OKX.com, we collect BTC/BCH/ETH data from 1/1/2020 to 4/22/2022 with total 1,212,300 lines.

Tick level limit order book data. Each tick order book data contains the timestamp, ask/bid price, ask/bid volume for each trading instrument. Asks consists of orders from other traders offering to sell an asset in this case. Bids are orders from traders offering to buy an asset. We record BTC/BCH/ETH's spot, futures and swap contract data from 4/19/2022 to 4/22/2022 and store data into local SQLite database file with total 2,791,623 lines.

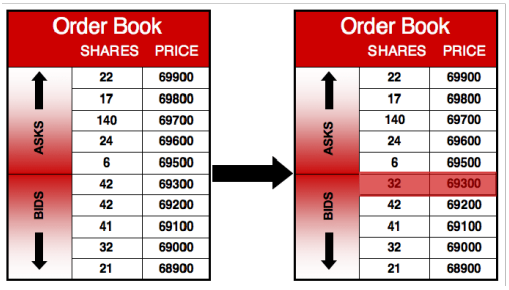

Figure 1. Example of limit order book bar data. [2]

3. Methods

3.1. Download minute bar data

We can collect data from OKX.com API *HTTP Request GET /api/v5/market/history-candles*. Rate limit set by ex-

change is 20 requests per 2 seconds. And maximum number of results per request is 100. There are total 1,212,300 lines of data during the time range. Therefore, theoretically, optimal time consumption is 1,212 seconds.

Optimization. We test the time consumption with BTC-USDT and improve it with multi-threading and multi-processing. The result shows multi-threading with 5 threads improves the most with 1,568 seconds. Larger thread number than 5 will make it exceed rate limit.

Method	# of Threads/Processes	Time Cost (s)
Theoretical optima		**1212**
No speed up		3846
Multi-threading	3	2628
Multi-threading	5	**1568**
Multi-processing	3	2811
Multi-processing	5	1705

Table 1. Time cost for download task.*The test is to download BTC-USDT contract minute bar data from 1/1/2020 to 4/22/2022. Multi-threading and multi-processing improve time cost significantly.

3.2. Record live tick data

We build an recorder function to register market data. Once new data is updated, market update event is triggered and data will be recorded in database. Since the time consumption only depends on the frequency of market data, single thread implementation satisfies our needs. SQLite database is used to store the live tick data.

3.3. Predictive Models

3.3.1 Approach

We predict the price spread of two highly correlated instrument such as BTC-spot/BTC-swap, ETH-spot/ETH-futures. Standard no-arbitrage pricing theory asserts that spot and futures prices must converge at expiration. Therefore, the two prices must converge. If not, there is an opportunity for arbitrage and risk-free profit. Our approach is to build features from tick data to predict the future price spread. Then, a pair trading rule can be made based on our prediction. If the price spread goes up a lot, spot's price would increase greater than swap's price and we can short spot and long swap to bet on their price spread reverting to their long-term mean.

3.3.2 Evaluation

We choose various indicators as our evaluation criteria, such as mean absolute error (MAE), mean square error (MSE), R square (R2), information coefficient (IC) and the overall running time. The most commonly used proxy for investment manager skill is the information ratio (IR). It

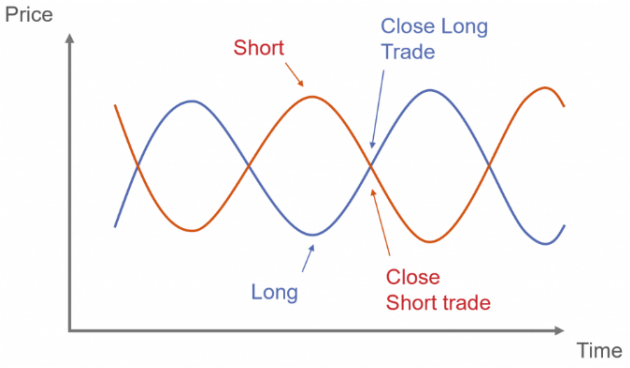

Figure 2. General idea of pair trading. [3]

is defined as the coefficient of variation of the manager's active returns [4]. In practice, people use derived information ratios as the product of the information coefficient and the breadth of an active management strategy. $IR = IC * Breath$. Breadth is number of independent "bets" taken per unit time. Information coeffcients (IC) describes the correlation between the actual and predicted stock returns that the managers considered in the investment management process. Because the predicted returns typically come from a stock selection model (or "alpha model"), information coefficient is also often used as a measure for the performance of such a model. The higher the information coefficient, the better performing the underlying stock selection model, and the more skillful the managers or model developers, suggests the conventional wisdom.

3.3.3 Parameters

Variables: The price ratio is defined as $PriceRatio = MidPrice(spot)/MidPrice(swap)$. It measures the price difference between spot and swap, and we use it as proxy for the price spread. We select the percentage change of 10, 20, 60 seconds of price ratio, relative spread, and order imbalance ratio as the predictors. The predicted variable is the percentage change of 30 ticks future return. A full list of explanatory and responding variables is shown in Table 2.

Notation	Name
x_1	$Relative\ spread\ spot$
x_2	$Relative\ spread\ swap$
x_3	$Order\ imbalance\ ratio\ spot$
x_4	$Order\ imbalance\ ratio\ swap$
x_5	$\Delta PriceRatio\ (\%) for\ the\ past\ 10S$
x_6	$\Delta PriceRatio\ (\%) for\ the\ past\ 20S$
x_7	$\Delta PriceRatio\ (\%) for\ the\ past\ 60S$
y	$(30\ Seconds) Change\ of\ future\ PriceRatio$

Table 2. Feature List

Train-test-split: We take the tick data from 4/19/2022 to 4/21/2022 as in-sample and 4/22/2022 as out-of-sample. This split ensure we can simulate intraday seasonality in the out-of-sample set.

3.3.4 Linear Models

Linear model is one of the most well known and commonly used algorithms in machine learning. It refers to techniques that linearly combine the input variables with some other constraints to develop output variable [5]. Representative models include Logistic Regression, Ridge Regression, Lasso Regression, etc.

Simple Linear Regression: First, we try out the simple linear regression model to predict the change of price spread. The estimator of coefficient takes the form [5]:

$$\beta = (X^T X)^{-1} X^T y$$

We use the scikit-learn package to implement the model. According to the results, $MSE = 1.044 \cdot 10^{-6}$, $IC = 0.489$, and $R^2 = 0.236$.

Ridge Regression: Since the features we select have some correlation, we also introduce Ridge regression as the solution to the possible multicollinearity issue. The estimator can be expressed by [5]:

$$\hat{\beta}_{ridge} = (X^T X + kI_p)^{-1} X^T y$$

where I_p is the $p \times p$ identity matrix, and k is a penalty coefficient of small value. Applying Ridge regression to predict the spread change, we have a slightly larger $MSE = 1.054 \cdot 10^{-6}$, greater information coefficient $IC = 0.513$, and slightly smaller $R^2 = 0.229$.

3.3.5 Ensemble Learning Models

Ensemble models consist of multiple alternative learners. It aims to obtain better predictive performance from a large amount of sub-models instead of developing single constituent algorithm alone [6]. By training sub-learners with or without the same model structure, the ensemble methods could be categorized into bagging and boosting algorithms. Generally, bagging generates predictions with lower variance, whereas boosting creates better predictions. Besides, the characteristics of ensemble algorithms allow us to apply parallel processing to increase the speed.

Bagging: Random Forest Random Forest is a classical bagging method, which constructs a brunch of decision trees and predicts by averaging the individual results [6]. We first choose max depth = 2, the model runs fast but with poor performance. The more trees are implemented, the higher accuracy the model will achieve. Finally we choose max depth = 5 with $R^2 = 0.261$ and running time 19.884 seconds for 7 cores.

Boosting: XGBoost Vs. LightGBM Two of the most popular algorithms that are based on Gradient Boosting are XGBoost and LightGBM (LGBM). Such methodology organizes different types of sub-learners based on the tree structure and combines their outputs to obtain better overall performance [7]. The main difference between the two algorithms is that XGBoost grows the tree via level-wise way, whereas LGBM uses a leaf-wise algorithm that results in more loss reduction [8]. Thus intuitively, XGBoost is capable of building more robust models. LGBM has a relatively higher accuracy and faster training speed, but it is easier to be overfitting.

Model summary: From Table 3, among all the models we build, XGBoost has the highest R-squared Score(0.291) and the highest Information Coefficient(0.539), followed by Random Forest(R^2 Score=0.261, IC=0.518), which implies that XGBoost makes the best prediction. Since the time cost of XGBoost(9.564 seconds for 1 job and 2.927 seconds for 7 jobs) is much smaller than Random Forest(101.95 seconds for 1 job and 19.884 seconds for 7 jobs), we set XGBoost as our final model in the end.

Model	# of Cores	R^2 Score	IC	Time (s)
Linear Reg	1	0.188	0.489	0.049
Ridge Reg	1	0.229	0.513	0.015
Random Forest	1	0.261	0.521	101.95
Random Forest	7	0.260	0.521	19.884
XGBoost	1	0.291	0.539	9.564
XGBoost	7	0.291	0.539	2.927
LGBM	1	0.132	0.386	1.022
LGBM	7	0.132	0.386	0.350

Table 3. Model Performance and Training Time.

3.3.6 Model Training Optimization

We improve the running speed by 413% for random forest and 226% for XGBoost with 7 cores. Although the performance doesn't improve until we increase the number of processes to 3, the possible reason behind is that sklearn uses joblib to execute tasks concurrently. It could induce a significant overhead because the input and output data need to be serialized in a queue for communication with the worker processes. [9]

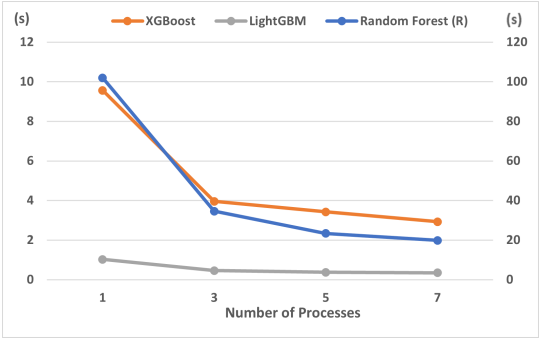

Figure 3. Running Time with Multiple Cores

3.4. Trading System Implementation

System structure. Here we build our trading system on other established open source codes. VeighNa [1] is a python-based open-sourced trading system used to trade Chinese futures market. We create our own applications including access to the OKX exchanges, data downloader, and trading strategy. The predictive models are loaded in the trading system so we can predict the future price movement and send orders to the market.

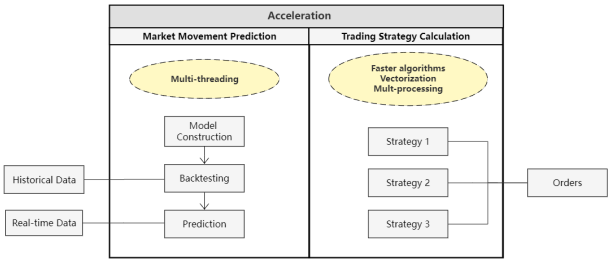

Figure 4. Structure of Trading System.

Trading rules. The trading rules is straightforward. The predictive models (XGBoost) are loaded at initialization. We can transform the live market data to features and then feed them to our models. Based on the prediction from the model (alpha), we can set a open limit parameter to decide when we will make the trades. Open limit 0.001 here means that when the alpha is larger then 0.001, we will send orders to long BTC spot and short BTC swap contracts.

Latency evaluation. We use Tick-to-Trade to measure the latency of the system. Tick-to-Trade [10] is the time interval between receiving a market tick data and processing the buy or sell order.

Implementation plans. We test several plans to implement the trading strategy part, which is the decisive factor for the latency. (1) Use pandas DataFrame to store a short period of live data and use pandas functions to calculate the features. (2) Use NumPy array to store a short period of live data and use NumPy functions to calculate the features. (3) Use improved algorithms and built-in data structures including deque and hashtables to store/update data. We find that the third plan has the boosted the efficiency most. The latency is almost all from the model prediction part where features are fed into a complex boosting tree model.

4. Conclusion and Future Work

We have proposed a real-time, low latency cryptocurrency trading system that operates with complex machine learning models. The latency from sending order to getting exchange response are at the level of 7ms. Our method also efficiently solve the data collection, data recording and model training steps and improved the speed with multithreading and multiprocessing techniques. For future works, we could implement more complicated models or use more performant languages like Cython or C++. We would also like to try the models and strategy on large-scale data or other pair of assets.

Model	Mean (ms)	Std (ms)
Plan 1: DataFrame	15.05	3.75
Plan 2: NumpyArray	10.33	3.07
Plan 3: Built-in data structures with improved algorithms	**7.05**	**2.06**
- Update variables	0.02	0.01
- Model predict	7.19	2.04
- Strategy and Send orders	0.04	0.02

Table 4. System Tick-to-Trade Latency.

References

[1] "vnpy," https://github.com/vnpy/vnpy, 2022. 1, 4

[2] C. S. Touré, ""continuum models for limit order books," 2016. 1

[3] Lucas Liew, "Pairs trading – a real-world guide," https://algotrading101.com/learn/pairs-trading-guide/, 2021. 2

[4] R. C. Grinold, "The fundamental law of active management," *Journal of Portfolio Management*, vol. 15, pp. 30–37, 1989. 2

[5] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. John Wiley & Sons, 2021. 3

[6] M. Belgiu and L. Drăguţ, "Random forest in remote sensing: A review of applications and future directions," *ISPRS journal of photogrammetry and remote sensing*, vol. 114, pp. 24–31, 2016. 3

[7] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794. 3

[8] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017. 3

[9] "Embarrassingly parallel for loops," https://joblib.readthedocs.io/en/latest/parallel.html#thread-based-parallelism-vs-process-based-parallelism, accessed: 2022-05-06. 3

[10] S. Rodgers, ""tick to trade" – is it the new must have metric in trading performance?" https://velocimetrics.com/tick-to-trade-is-it-the-new-must-have-metric-in-trading-performance, 2018. 4