OKX High Performance Trade Simulator

1. Overview

Building a real-time trade simulator that estimates slippage, fees, market impact, transaction cost, maker/taker proportion, latency using full-depth L2 orderbook data from OKX.

2. Key Features

- Live Feed

The system connects to given real-time WebSocket stream.

It updates and processes the data with each tick.

The data is temporarily stored in Redis for fast access and enables metric updates based on the most current market data.

- Slippage Model

It estimates the cost incurred due to the difference in expected price and execution price. Implemented a linear regression model and predicted using top bid and top ask price.

$$\text{Slippage} = C0 + C1\ (Q/V)$$

- Fees Model

Computes estimated trading fees on selected fee trier.

$$\text{Fee} = \text{Trade Amount} * \text{Fee Rate}$$

- Market Impact

Implements the Almgren-Chriss model to estimate the price movement caused by executing large orders.

impact = eta * (Q / V) + 0.5 * lam * (σ ** 2) * (Q / V) ** 2

where Q= order size in USD, V= daily volume, sigma = volatility

- Maker/Taker Proportion

Used logistic regression to predict likelihood of an order being maker or taker…..i.e. whether the next trade is passive or aggressive.

The ratio helps assess market behaviour bias.

P (maker) = 1/1+exp(-(B0+B1x1+…..Bnxn))

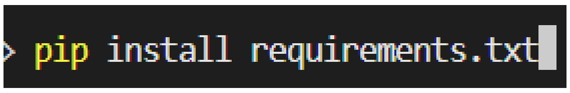
P (taker) = 1- P(maker)

- Latency

Local Machine Timestamp – Execution Timestamp

3. Getting Started

1) Python Dependencies



```
> pip install requirements.txt
```

2) Memurai
   As Redis doesn’t officially support Windows, we will need Memurai.
   Install Free Developer Edition from https://www.memurai.com/get-memurai
   And then run it as a Windows Service

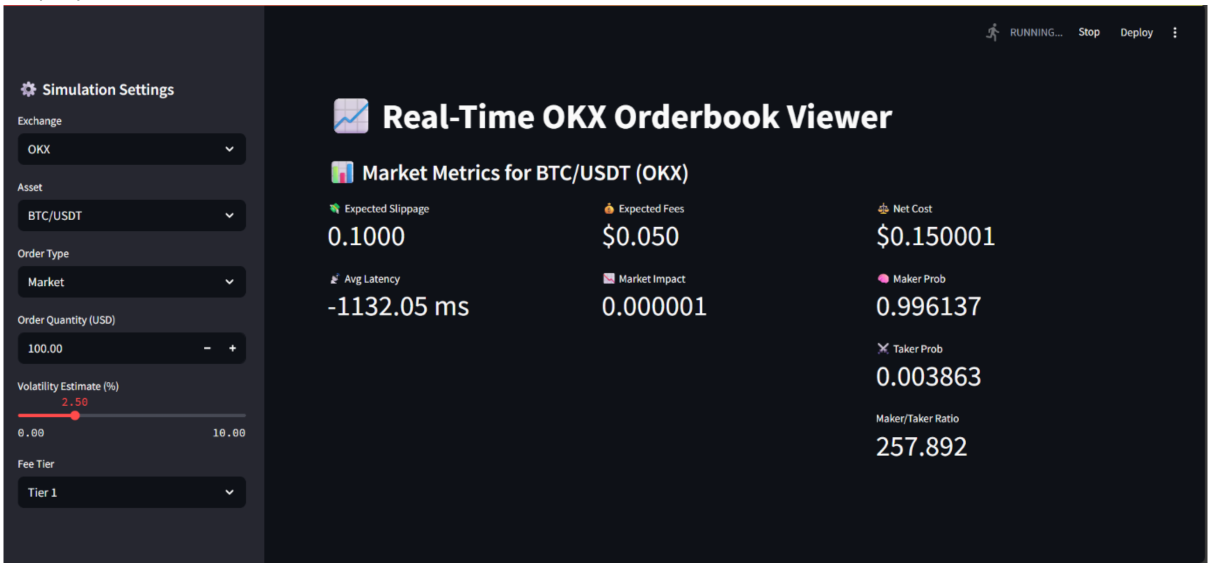
4. System Architecture

5. Data Collection

1) Websocket URL
   wss://ws.gomarket-cpp.goquant.io/ws/l2-orderbook/okx/BTC-USDT-SWAP
2) Data Storage
   a) CSV
      To be quite honest, real time data processing was new for me so first I tried to do it the old fashioned way. I coded the websocket terminal to get back the data in form of timestamp, bid price, ask price, latency, which I stored as CSV file.
   b) System Memory Queue
      But soon I realized using CSV file, I won’t be able to calculate the params real time therefore, I created a deque data structure to hold my data extracted from OKX L2.
   c) Redis
      But Python modules don’t let models to share memory across processes..ie backend and frontend couldn’t process the data together. That’s why I had to switch my approach to Redis which is a in-memory key-value database which allows shared memory model.

6. User Interface

I’ve built a clean and interactive Streamlit-based dashboard for simulating trade in real time. It is divided into two major components: sidebar input controls and a dynamic output display.



Simulation Settings

Exchange: OKX

Asset: BTC/USDT

Order Type: Market

Order Quantity (USD): 100.00

Volatility Estimate (%): 2.50 (0.00 – 10.00)

Fee Tier: Tier 1

Real-Time OKX Orderbook Viewer

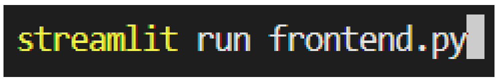
Market Metrics for BTC/USDT (OKX)

Expected Slippage	Expected Fees	Net Cost
0.1000	$0.050	$0.150001
Avg Latency	Market Impact	Maker Prob
-1132.05 ms	0.000001	0.996137
		Taker Prob
		0.003863
		Maker/Taker Ratio
		257.892

The app uses st.empty() to allow full panel refresh on every tick without reinitializing the entire UI where Redis serves as the communication bridge between the WebSocket backend and frontend.

7. Running

Unzip the compressed file.
Install all the right dependencies.
Run the backend.py file in Python environment.
Run the frontend.py file in Streamlit.



```
streamlit run frontend.py
```

8. File Structure

```
GOQUANT/
 ├── .venv/
 |
 ├── docs/
 |   └── Draft1.docx
 |
 ├── models/
 |    ├── __pycache__/
 |    ├── fees.py
 |    ├── maker_taker.py
 |    ├── market_impact.py
 |   └── slippage.py
 |
 ├── utils/
 |    ├── backend.py
 |   └── frontend.py
 |
 └── requirements.txt
```

9. Acknowledgements

I would like to firstly thank the recruitment team behind GoQuant, for providing a well structured and intellectually stimulating assignment that reflects real-world trading system challenges.

Then, I would like to thank the open source libraries – including pandas, streamlit, redis and scikit-learn for enabling rapid prototyping and model integration.

Also I would extend my gratitude to:-

a) Python Community
b) OKX Exchange – for offering public WebSocket access to real-time L2 market data.
c) Memurai – for enabling Redis support on Windows environment for development.