Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com

Veloz: High Performance Trading Simulator

- By Yashraj Sakunde (*yashrajsakunde@gmail.com*, *Linkedin*)

1. Objective:

Create a high-performance trade simulator leveraging real-time market data to estimate transaction costs and market impact. This system will connect to provided WebSocket endpoints that stream full L2 orderbook data for cryptocurrency exchanges.

2. Scope:

- Real-time market data processing for all assets under OKX using OKX public API and OKX and Binance API for scraping historical datasets
- Implementation of various models for estimation of parameters including slippage, market impact, net cost, etc
- UI for visualization and interaction
- Performance and metrics logging
- Model training for data sets

3. System Requirements:

3.1 Hardware Requirements:

- Minimum: Intel i5, 8GB RAM, Stable internet
- Recommended: Intel i7+, 16GB+ RAM, Low-latency network

3.2 Software Requirements

- Python 3.10+
- PyQt5
- NumPy, scikit-learn
- Pandas
- websockets, requests
- json
- (Packages and dependencies included in environment.yml file)

4. Setup and Installation requirements:

1. The simulator is built using python in conda environment.
2. Install Anaconda or Miniconda on your pc through browser.
3. Extract the zip file and locate the files on appropriate location on your pc.
4. Setup conda environment by opening the terminal or Anaconda prompt and navigating to the path to the folder named “Trading_simulator”, using :
   ```
   cd path/to/simulatorfolder    (replace with actual path)
   ```
5. Run the following command to create the environment:
   ```
   conda env create -f environment.yml
   ```
6. Activate the environment using:
   ```
   conda activate base (“base” is the name written in environment.yml file on the first line)
   ```
7. Install all the required packages mentioned in yml file manually in case of missing packages.

Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com

8. Run the command:
   python main.py (main.py is the entry point)

NOTE: Preferred to setup and run on MacOS for demo purpose

5. File structure and description:

```
/Veloz
 |
 |- main.py
 |- ui.py
 |- websocket_client.py
 |- slippage_history.csv          (obtained after running build_slippage_history.py in standalone)
 |- maker_taker_history.csv       (obtained after running build_maker_history.py py in standalone)
 |- environment.yml
 |- pip_requirements.txt
 |
 |-- /models
          |- impact.py
          |- maker_taker.py
          |- slippage.py
          |- train_slippage_model.py
          |- train_maker_taker.py
          |- __init__.py
          |- slippage_model.pkl        (obtained after running train_slippage_model.py in standalone)
          |- maker_taker_model.pkl     (obtained after running train_maker_taker.py in standalone)
 |-- /utils
          |- __init__.py
          |- fees.py
          |- latency.py
|-- /scripts
          |- build_slippage_history.py
          |- build_maker_history.py
|-- /logs
          |- latency_metrics.log
```

6. Models and Techniques Used:

6.1 Linear Regression:

6.11 Purpose:

To estimate slippage by quantifying market factors and execution costs

6.12 Why linear regression model?

- Provides a simple and straight forward mathematical framework to model the relationship between slippage and influencing factors such as order size, volatility, and bid-ask spread. The coefficients derived from the model offer clear insights into how each factor contributes to slippage, facilitating better decision-making.
- Linear regression models are computationally efficient, allowing for rapid estimation and real-time adjustments.
- Linear regression serves as a foundational tool that can be extended to more complex models if needed. It allows traders to start with a basic model and incorporate additional variables or switch to more sophisticated techniques as required.

6.13 Concept:

Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com

- Key-parameters:
- **Order size/ average daily volume (USD)** => captures liquidity impact, so more the order size, more could be the slippage.
Coefficient **β1** gives slippage increase per unit rise in size/avg. daily volume.

- **Volatility** => given by standard deviations of returns.
High volatility means rapid price movements and more execution uncertainty.
Coefficient **β2** measures how slippage is affected by market instability.

- **Bid-ask spread** => Spread = Ask price - bid price/ Mid price x 100.
Indicates difference between highest price buyer is willing to pay and lowest price seller is willing to accept.
More the spread, lesser the liquidity and higher the slippage.
Coefficient **β3** reflects cost of immediate execution.

- **Momentum** => Indicates slope of price trend.
Trading against momentum can increase slippage.
Coefficient **β4** quantifies penalty for contrarian trades

Model Equation:

Slippage (bps) = β0 + β1(ADVSize) + β2(Volatility) + β3(Spread) + β4(Momentum) + ϵ

6.14 Implementation:
1. build_slippage_history.py file is created which collects and logs trade data from OKX cryptocurrency exchange, specifically for BTC-USDT trading pair (for demo). It fetches historical trade data, current order book snapshots, and 24-hour trading volume, then computes relevant metrics like mid-price, spread, and market depth. The collected data is written to a CSV file for further analysis.

2. train_slippage_model.py file is created to train the linear regression model from the data fetched into csv file so that a basic estimate can be made to indicate slippage behavior, using Ordinary Least Squares (OLS) model:

$$\min_{\beta} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}))^2$$

The evaluation metrics here are:
Mean Absolute Error (MAE): Average absolute difference between predicted and actual slippage values. Used to assess accuracy of the regression model.
R^2 Score: Proportion of variance in the dependent variable predictable from the independent variables.
Cross-Validation R^2: Average R^2 score across 5-fold cross-validation, providing insight into the model's generalizability.

- joblib is used to save the data to a file in .pkl format.

Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com

3. slippage.py is used to load the serialized model from slippage_model.pkl to further predict expected slippage.

References:
https://andrewilardi.com/2017/12/22/slippage-root-mean-squared-error-model-performance/
https://trendspider.com/learning-center/slippage-in-trading-understanding-the-invisible-impact-on-your-trades/
https://fastercapital.com/content/Slippage-and-Spread--Unveiling-the-Relationship-for-Better-Trades.html
https://builtin.com/data-science/ols-regression

6.2 Logistic Regression:

6.21 Purpose:

Used to predict the binary outcome i.e whether it is a maker (liquidity provider) or taker(liquidity consumer) so that fees could be charged accordingly.

6.22 Why Logistic Regression?

Unlike simple linear regression, which is used to predict continuous values, logistic regression is moreover used to tell two outcomes like binary, for example whether something is true or false. This behavior of logistic regression helps in deciding maker and taker in trading so that the fees could be charged according to the liquidity provision or consumption of the participating entity.

6.23 Concept:

Makers => Traders who place limit orders (e.g., "Buy BTC at $50,000") that add liquidity to the order book. They receive rebates.
Takers => Traders who execute market orders (e.g., "Buy BTC now at market price") that remove liquidity. They pay fees.

We get a probabilistic outcome:

$$P(y=1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)}}$$

Where,
P(y=1): Probability of being a maker order
βi : Coefficients quantifying feature impact on log-odds
xi : Predictive features (e.g., order size, spread)

6.24 Implementation:

1. build_maker_taker_history.py collects real-time trade and order book data from Binance's API for the BTC-USDT trading pair and writes it into a CSV file for later analysis or training a machine learning model. It fetches most recent historical trade from binance. Fetches top 5 bid and ask orders and calculates bid and ask depth(volume) for the top 5 levels.

2. train_maker_taker.py trains a logistic regression model to predict whether a crypto trade was made by a taker (someone who removes liquidity) based on order book and

Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com

trade features derived from your CSV (maker_taker_history.csv). Computes aggressiveness.
Aggressiveness = how far into the spread the trade executed:
For buys: trade price - best_bid
For sells: best_ask - trade price

- A **positive aggressiveness value** indicates that the trade was executed at a price less favorable than the best available, suggesting an aggressive trade.
- A **zero or negative value** suggests a passive trade, executed at or better than the best available price.

Compares trade size to liquidity in top 5 order book levels.
Outputs precision, recall, F1-score, and ROC AUC (how well the model distinguishes takers from makers).

1. **Precision:**
   The ratio of correctly predicted positive observations to the total predicted positive observations.It answers: *Of all trades predicted as aggressive, how many were truly aggressive?*

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

2. **Recall:**
   The ratio of correctly predicted positive observations to all actual positive observatio ns. It answers: *Of all actual aggressive trades, how many did the model correctly ide ntify?*

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

3. **F1-Score:**
   The harmonic mean of precision and recall, providing a balance between the two. It's especially useful when the class distribution is uneven.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

4. **ROC AUC (Receiver Operating Characteristic - Area Under Curve):**
   Measures the model's ability to distinguish between classes across all thresholds. The ROC curve plots the true positive rate against the false positive rate.

Interpretation:

- An AUC of **1.0** indicates perfect distinction between classes.
- An AUC of **0.5** suggests no discriminative power, equivalent to random guessing.

Saves the trained model to a .pkl file so it can be used later for predictions.

3. makertaker.py used to predict probability of maker or taker of current trades by applying logistic regression while referencing the previously trained model.

Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com

References:
https://academy.binance.com/en/articles/what-are-makers-and-takers
https://web.stanford.edu/class/msande448/2017/Final/Reports/gr4.pdf
https://www.webmobinfo.ch/blog/market-making-algorithms-how-they-work

6.3 Gatheral's Non-Linear Market Impact Model:

6.31 Purpose :

To estimate non-linear market impact costs during trade execution by capturing empirical observations of concave (non-linear) impact behaviour To estimate non-linear market impact costs during trade execution by capturing empirical observations of concave (non-linear) impact behaviour.

6.32 Ideal Model:

The **Gatheral's Non-Linear Market Impact Model** enhances realism in market impact modelling by using a power-law function to reflect how impact scales with trade size. It separates temporary and permanent impact with empirically backed non-linear relationships.

Key parameters:

- β (beta): Non-linearity coefficient for temporary impact (typically 0.5 < β < 1)
- η (eta): Permanent impact coefficient
- σ (sigma): Asset volatility
- λ (lambda): Risk aversion level (optional, for extensions involving risk minimization)
- Q: Total quantity to be traded
- T: Total trading time
- v: Participation rate or trading speed
- V: Daily traded volume

The model separates market impact into:

- **Temporary impact** (I_t):

$$I_t = k\left(\frac{q}{V}\right)^{\beta}$$

where q is the trade size and V is the daily volume. This represents the concave (diminishing) cost per unit as volume increases.

- **Permanent impact** (I_p):

$$I_p = \eta \cdot \frac{Q}{V}$$

This captures the sustained price movement after trade execution due to informational effects.

Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com

Implementation involves fitting historical trade data to determine β and other constants using regression techniques. This model does not necessarily yield closed-form solutions but provides **more realistic inputs** to execution algorithms, especially in high-volume or algorithmic trading strategies.

6.33 Model Implemented:

The model implements **Gatheral's Non-Linear Transient Impact Model**, incorporating concave market impact and time-decaying influence of trades.

- **Mid-price** is estimated from top order book levels, and USD notional is converted to shares.
- The trading horizon is split into discrete intervals.
- A **power-law decay kernel** models how past trades affect current price.
- Optimal trading rates are computed using **fixed-point iteration**, balancing non-linear impact (δδ), decay (γγ), and risk aversion (λλ).
- Final cost includes **transient impact**, **permanent impact**, and a **risk term**, all converted to USD.

This approximation captures the essential dynamics of Gatheral's model while remaining computationally efficient.

7. Optimization:

7.1 Memory Management:

- **Rolling Window Buffers:** The use of fixed-length lists (e.g., self._arrival_times, self._proc_times, self._ui_times) with manual pop operations ensures that memory usage remains bounded. By limiting the window size (e.g., 100), the application prevents unbounded memory growth, which is crucial for long-running trading systems.
- **Selective Data Retention:** Only essential metrics and recent order book snapshots are kept, reducing memory footprint and avoiding unnecessary data accumulation.

7.2 Network Communication:

- **WebSocket Protocol:** The system uses asynchronous WebSocket connections, which are more efficient than repeated HTTP polling for real-time data. This reduces network overhead and latency, ensuring timely order book updates.
- **Timeout Handling:** The use of asyncio.wait_for(ws.recv(), timeout=5) ensures the client does not hang indefinitely, improving robustness and allowing for quick recovery from network issues.
- **Error Logging and Reconnection:** Exceptions are caught and logged, and the client attempts to reconnect, which helps maintain persistent connectivity and minimizes downtime.

7.3 Data structure selection:

- **List Comprehensions for Order Book Parsing:** The code uses list comprehensions (e.g., [float(level) for level in orderbook[:10]]) for efficient and concise extraction of price levels from the order book, which is both readable and performant for small, fixed-size slices.
- **Windowed Metrics:** By keeping only the most recent N samples (e.g., window size of 100), the system avoids unnecessary storage of historical data, which is not needed for real-time analytics.

Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com

7.4 Thread management:

- **QThread for Asynchronous Processing**: The WebSocket client runs in a separate QThread, ensuring that network and data processing do not block the main GUI thread. This design maintains UI responsiveness and allows for smooth user interactions, even during intensive real-time data processing.
- **Signal-Slot Mechanism:** The use of Qt's pyqtSignal for communication between threads ensures thread-safe updates to the UI, preventing race conditions and crashes.

7.5 Regression model efficiency:

- **Modular Model Functions:** Functions like calculate_impact, estimate_slippage, and predict_maker_taker are imported as modular components. This separation allows for efficient, targeted updates and testing of each model without affecting the overall system.
- **On-Demand Calculations:** Regression and impact calculations are performed only when new data arrives (i.e., on each tick), avoiding unnecessary recomputation and reducing CPU load.
- **Simple, Interpretable Models:** The use of straightforward, interpretable regression models (as seen in your calculate_impact and slippage estimation) ensures fast computation, which is critical for real-time trading environments.

~ I found better optimization techniques in some areas like:

1. *Using numpy arrays instead of python lists to make operations faster*
2. *Using Deque which has complexity of O(1) for pop/append operations unlike lists which has O(n) complexity.*
3. *Binary protocol buffers to reduce message size*

But was unaware of these at first and implementation would have required more time.

8. Latency Logging (Bonus section) :

- Data processing latency
- UI update latency
- End-to-end simulation loop latency

These parameters are logged into log file which is stored in the logs folder.

9. Concepts and Terminologies :

9.1 Trading and market terms:

1. ***Spot Trading:*** *Buying or selling assets for immediate delivery at the current market price.*

2. ***Order Type:*** *Instructions for executing trades, such as market orders (immediate execution at current price), limit orders (execution at a specified price or better), and stop orders (execution once a certain price is reached).*

3. ***Market Type:*** *Classification of trading environments, e.g., spot markets (immediate settlement) or derivatives markets (futures, options).*

4. ***Fee Tier:*** *A pricing structure where trading fees vary based on factors like trading volume or user status.*

5. ***Volatility:*** *A statistical measure of the dispersion of returns for a given asset, indicating the degree of variation in its trading price over time.*

Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com

6. ***Market Impact:*** *The effect that a trade has on the market price of an asset, often significant with large orders.*

7. ***Market Depth:*** *The market's ability to sustain large orders without significant impact on the price, reflecting the volume of buy and sell orders at various price levels.*

8. ***Order Book:*** *A real-time list of buy and sell orders for a specific asset, organized by price level.*

9. ***Bid Depth:*** *The total quantity of buy orders at various price levels below the current market price.*

10. ***Ask Depth:*** *The total quantity of sell orders at various price levels above the current market price.*

11. ***Maker:*** *A trader who provides liquidity by placing limit orders that are not immediately filled.*

12. ***Taker:*** *A trader who removes liquidity by placing orders that are immediately matched with existing orders.*
13. ***Liquidity:*** *The ease with which an asset can be bought or sold in the market without affecting its price.*

14. ***Market Spread:*** *The difference between the highest bid price and the lowest ask price in the market.*

15. ***Aggressiveness:*** *The degree to which a trader is willing to buy or sell at prices away from the current market price, often leading to immediate execution.*

16. ***Slippage:*** *The difference between the expected price of a trade and the actual price at which the trade is executed.*

17. ***Mid Price:*** *The average of the current bid and ask prices for an asset*

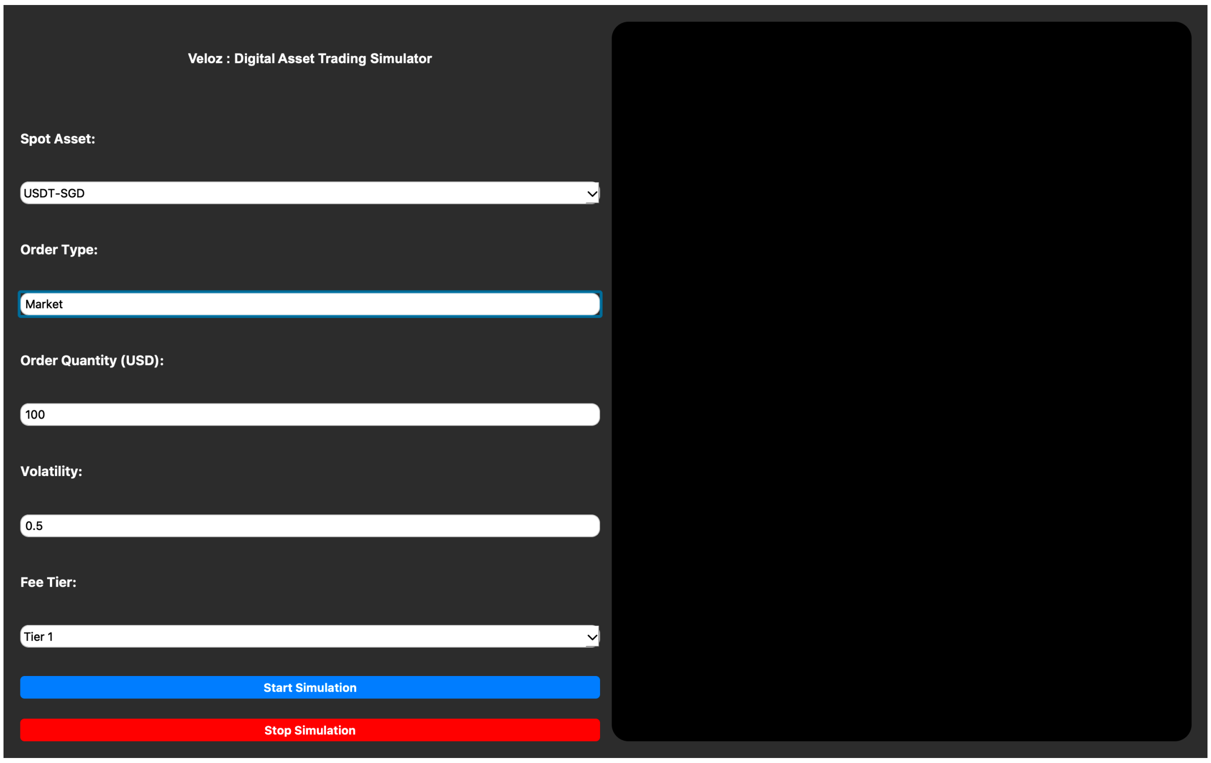
9.2 Statistical and machine learning terms:

1. ***Mean Absolute Error (MAE):*** *A measure of prediction accuracy in a model, calculated as the average absolute difference between predicted and actual values.*

2. ***Cross Validation:*** *A technique for assessing how a predictive model will perform on an independent dataset, by partitioning the data into subsets, training the model on some subsets and validating it on others.*

Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com

10. UI of Simulator:



Veloz : Digital Asset Trading Simulator

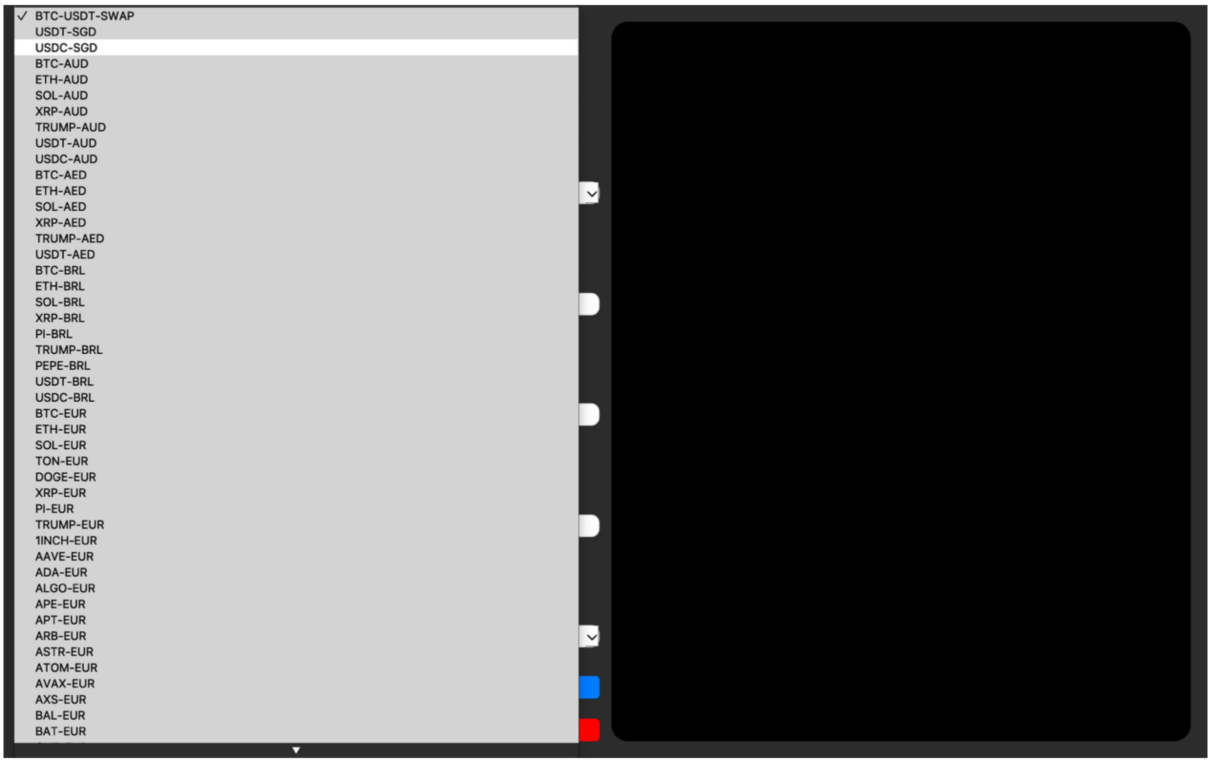
Spot Asset: USDT-SGD

Order Type: Market

Order Quantity (USD): 100

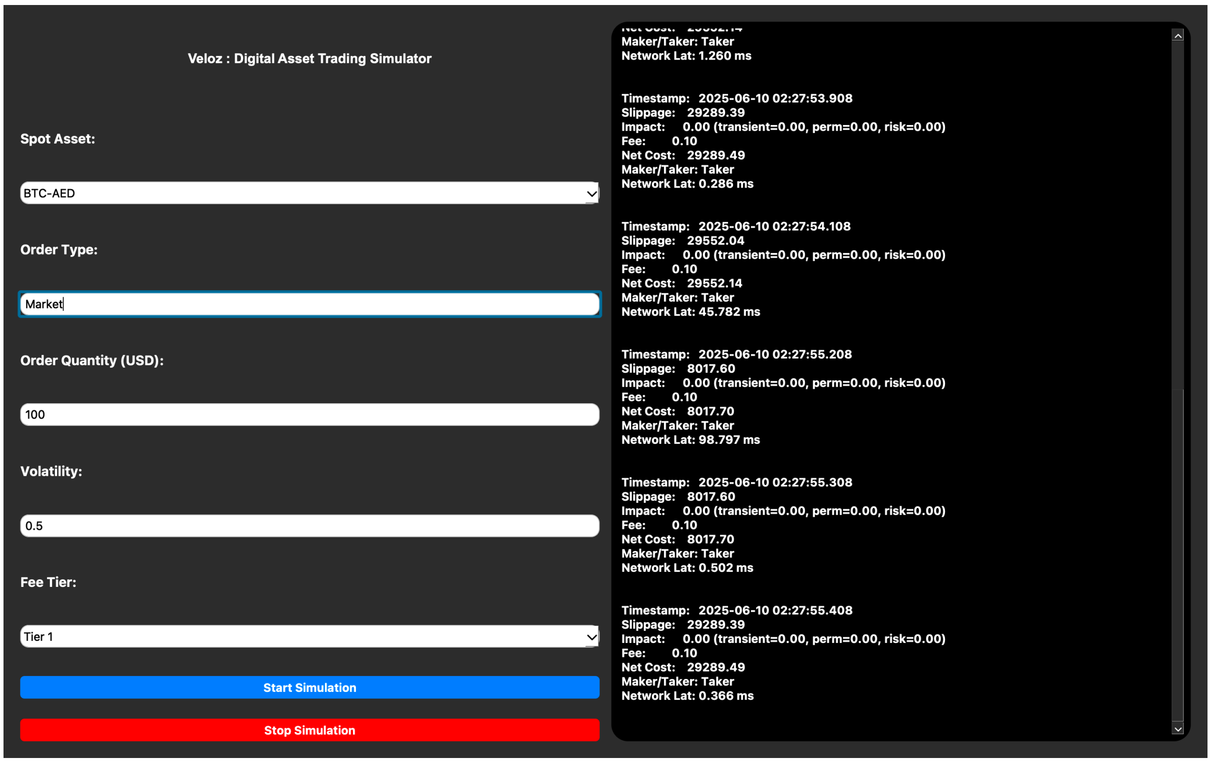
Volatility: 0.5

Fee Tier: Tier 1



✓ BTC-USDT-SWAP
USDT-SGD
USDC-SGD
BTC-AUD
ETH-AUD
SOL-AUD
XRP-AUD
TRUMP-AUD
USDT-AUD
USDC-AUD
BTC-AED
ETH-AED
SOL-AED
XRP-AED
TRUMP-AED
USDT-AED
BTC-BRL
ETH-BRL
SOL-BRL
XRP-BRL
PI-BRL
TRUMP-BRL
PEPE-BRL
USDT-BRL
USDC-BRL
BTC-EUR
ETH-EUR
SOL-EUR
TON-EUR
DOGE-EUR
XRP-EUR
PI-EUR
TRUMP-EUR
1INCH-EUR
AAVE-EUR
ADA-EUR
ALGO-EUR
APE-EUR
APT-EUR
ARB-EUR
ASTR-EUR
ATOM-EUR
AVAX-EUR
AXS-EUR
BAL-EUR
BAT-EUR

Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com



Veloz : Digital Asset Trading Simulator

Spot Asset: BTC-AED

Order Type: Market

Order Quantity (USD): 100

Volatility: 0.5

Fee Tier: Tier 1

```
Maker/Taker: Taker
Network Lat: 1.260 ms

Timestamp:  2025-06-10 02:27:53.908
Slippage:   29289.39
Impact:     0.00 (transient=0.00, perm=0.00, risk=0.00)
Fee:        0.10
Net Cost:   29289.49
Maker/Taker: Taker
Network Lat: 0.286 ms

Timestamp:  2025-06-10 02:27:54.108
Slippage:   29552.04
Impact:     0.00 (transient=0.00, perm=0.00, risk=0.00)
Fee:        0.10
Net Cost:   29552.14
Maker/Taker: Taker
Network Lat: 45.782 ms

Timestamp:  2025-06-10 02:27:55.208
Slippage:   8017.60
Impact:     0.00 (transient=0.00, perm=0.00, risk=0.00)
Fee:        0.10
Net Cost:   8017.70
Maker/Taker: Taker
Network Lat: 98.797 ms

Timestamp:  2025-06-10 02:27:55.308
Slippage:   8017.60
Impact:     0.00 (transient=0.00, perm=0.00, risk=0.00)
Fee:        0.10
Net Cost:   8017.70
Maker/Taker: Taker
Network Lat: 0.502 ms

Timestamp:  2025-06-10 02:27:55.408
Slippage:   29289.39
Impact:     0.00 (transient=0.00, perm=0.00, risk=0.00)
Fee:        0.10
Net Cost:   29289.49
Maker/Taker: Taker
Network Lat: 0.366 ms
```

Name: Yashraj Sakunde
Email: yashrajsakunde@gmail.com
Linkedin