# **GoQuant\_Simulator**

## 

## **Overview**

**GoQuant Simulator** is a real-time modular trading cost simulator designed to estimate execution costs such as **slippage**, **market impact**, and **maker-taker fee proportions**. It integrates live L2 orderbook data from OKX using a WebSocket feed and provides continuous simulation updates to enable fast and interactive trading cost estimation based on actual market conditions.

## **🔧 Project Architecture**

GoQuant\_Simulator/

├── backend/ # FastAPI backend with core logic

│ ├── main.py # FastAPI application entry

│ └── services/

│ ├── websocket\_client.py # Real-time L2 orderbook connector

│ └── orderbook\_processor.py # Orderbook snapshot handler

│ └── models/

│ ├── simulation.py # Orchestration logic for simulations

│ ├── frontenddata.py # Input schema using Pydantic

│ ├── slippage.py # Slippage prediction model

│ ├── maker\_taker.py # Maker-taker prediction model

│ └── market\_impact.py # Almgren–Chriss cost function

├── frontend/ # React frontend built with Vite

├── GOquant\_Simulator.ipynb # Jupyter notebook for prototyping

├── README.md

└── requirements.txt # Python dependencies

## 

## 

## 

## 

## **Features**

* Real-time Orderbook Feed: Live L2 data from GoQuant WebSocket (OKX BTC-USDT-SWAP)
* Deep Learning Models: Neural networks estimate slippage and maker-taker fill proportions
* Market Impact Modeling: Almgren–Chriss quadratic function
* Interactive Simulations: Customize inputs such as trade size, volatility, and fee tier
* Cost Breakdown Output: Slippage, fees, market impact, net cost, and latency in real time

## **Tech Stack**

| **Component** | **Technology** |
| --- | --- |
| Backend | FastAPI, WebSocket, Pydantic |
| DL Framework | TensorFlow, Keras |
| Frontend | React.js, Vite, TailwindCSS |
| Data Source | GoQuant L2 Orderbook (OKX) |

## **Models**

### **1. Slippage Model**

## Objective: Estimate the price deviation caused by executing a trade of a given size under current market liquidity and volatility.

## Inputs:

## Trade size (USD)

## Volatility

## Market liquidity (derived from L2 snapshot)

## Output: Predicted slippage cost (in percentage or basis points)

## Architecture:

## Fully Connected Feedforward Neural Network

## Activation: ReLU, Output: Linear

## Trained using synthetic data generated with Gaussian noise

## Implementation: SlippageModel class (Keras Sequential API)

## Training Strategy:

## Synthetic data creation with domain-specific assumptions

## Loss: MSE

## Early stopping based on validation loss

### **2. Maker-Taker Model**

## Objective: Predict the proportion of trade filled as a maker (adds liquidity) vs taker (removes liquidity).

## Inputs:

## Normalized trade size

## Normalized volatility

## Normalized liquidity

## Output: Float between 0 and 1 (probability of being a maker)

## Architecture:

## Dense layers with batch normalization and dropout

## Sigmoid output for probability

## Assumptions:

## Larger and faster trades are more likely to be taker orders

## Higher liquidity environments favor maker fills

## Implementation: MakerTakerModel class (Keras Functional API)

### **3. Market Impact Model**

## Objective: Estimate permanent market price impact due to large trade execution

## Formula:

## impact = alpha \* trade\_size + beta \* trade\_size \*\* 2

## Inputs:

## Trade size (USD)

## Outputs:

## Market impact cost (USD)

## Parameters:

## alpha: Linear coefficient

## beta: Quadratic coefficient

## Model Type: Almgren-Chriss-style quadratic cost function

## **Frontend**

The React + Vite-based frontend allows users to:

* Input simulation parameters (order type, quantity, side, volatility, fee tier)
* Visualize continuous updates of cost metrics
* View granular cost breakdowns
* Interact with a real-time simulation interface

## **Backend**

## **Startup Tasks**

### **@app.on\_event("startup") :** This function runs automatically when the FastAPI server starts. It performs critical initializations required for simulation:

* Initializes Models
  + maker\_taker\_model = MakerTakerModel() – Prepares the maker-taker cost estimation model.
  + slippage\_model = SlippageModel() – Prepares the slippage estimation model.
* Starts Background WebSocket Listener
  + asyncio.create\_task(connect\_to\_websocket()) – Begins listening to live orderbook data (e.g., from OKX or another exchange) in the background.

This ensures that all required components are ready before any client WebSocket connection attempts simulation.

**API Endpoints**

### **1. GET/ :** Returns a simple message to indicate the backend is running.

## Response

{

"message": "GoQuant Simulator Backend"

}

### **2. WebSocket /ws/simulate** : Accepts a JSON object with simulation parameters and returns updated cost estimates every second.

#### **Input Schema (via WebSocket message)**

{

"exchange": "OKX",

"spotAsset": "BTC-USDT",

"orderType": "market",

"quantity": 10000,

"volatility": 0.05,

"feeTier": "Tier 2",

"marketSide": "buy"

}

**Output Schema (streamed every second)**

{

"slippage": 0.0013,

"fees": 0.35,

"marketImpact": 0.58,

"netCost": 0.9313,

"makerTakerRatio": 0.6247,

"internalLatency": 0.0124

}

## **WebSocket Data Feed**

* URL:wss://ws.gomarket-cpp.goquant.io/ws/l2-orderbook/okx/BTC-USDT-SWAP
* Function: Real-time Level-2 orderbook updates.
* Implementation: services/websocket\_client.py
* Snapshot Maintainer: orderbook\_processor.py
* Features:
  + Auto-reconnect logic
  + In-memory orderbook
  + Update frequency: ~1 second

#### **WebSocket Internals**

## Validates the input using FrontendData **Pydantic** model.

## Uses get\_orderbook\_snapshot() to get current bids/asks.

## Computes cost metrics using:

## SlippageModel

## MakerTakerModel

## Almgren-Chriss quadratic model

## Sends updated simulation every second until disconnection.

## **References**

## FastAPI – Modern, Fast (High-performance) web framework for building APIs with Python https://fastapi.tiangolo.com

## WebSockets in FastAPI https://fastapi.tiangolo.com/advanced/websockets

## AsyncIO – Asynchronous I/O in Python <https://docs.python.org/3/library/asyncio.html>

## Pydantic – Data validation and settings management using Python type annotations https://docs.pydantic.dev

## Almgren–Chriss Model for Optimal Execution Almgren, R., & Chriss, N. (2000). *Optimal execution of portfolio transactions*. Journal of Risk, 3(2), 5–39. https://papers.ssrn.com/sol3/papers.cfm?abstract\_id=1108725

## François Chollet, *Deep Learning with Python* (Keras) [https://keras.io](https://keras.io/)

## React – A JavaScript library for building user interfaces [https://react.dev](https://react.dev/)

## Native WebSocket: <https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>

## React WebSocket Hook: <https://www.npmjs.com/package/react-use-websocket>

## Tailwind CSS [https://tailwindcss.com](https://tailwindcss.com/)

## 

## 

## 

## 

## 

## 

## 

## 