Trade Simulator: Technical Documentation

Table of Contents

Project Overview

The Trade Simulator is a high-performance, real-time market data simulator for cryptocurrency trading. It estimates transaction costs, market impact, and slippage using live L2 orderbook data from exchanges (e.g., Binance). The system is designed for research, strategy backtesting, and educational purposes.

System Architecture

The Trade Simulator is built with a modular architecture consisting of several key components:

- **Frontend:** PyQt6/PySide6-based GUI with light/dark themes and interactive pyqtgraph charts
- **Backend:** Python async engine for WebSocket data ingestion, model computation, and performance monitoring
- **Models:** Modular classes for market impact, slippage, and maker/taker prediction
- **Logging:** Centralized logging for debugging and performance analysis
- **Data Visualization:** Real-time charting system for price, depth, and metrics
-

System Architecture Diagram

```
┌─────────────────────────────────────────────────────────────┐
│                       Trade Simulator                         │
└────────────────────────────┬────────────────────────────────┘
                             │
         ┌───────────────────┴──────────────────┐
         ▼                                       ▼
```

```
┌─────────────────┐                  ┌─────────────────┐
│    UI Layer     │                  │  WebSocket      │
│  (PySide6/Qt)   │◄────────────────►│ Client          │◄─────► Crypto Exchange
└─────────────────┘                  └─────────────────┘          APIs
        │                                    │
        ▼                                    ▼
┌─────────────────┐                  ┌─────────────────┐
│  Trade Models   │                  │  Utility        │
│  & Calculations │                  │  Services       │
└─────────────────┘                  └─────────────────┘
```

Data Flow:

1. WebSocket client connects to the exchange and receives L2 orderbook updates
2. Data is processed, validated, and stored in history buffers
3. Models compute slippage, impact, and probabilities
4. UI updates outputs, orderbook preview, and real-time charts
5. Performance metrics are logged, displayed, and charted

User Interface (UI/UX)

UI Features

- Themed, responsive layout with PyQt6/PySide6
- Gradient header bar with app name
- Tabbed interface with simulator and analytics sections
- Input Panel: Exchange, asset, order type, quantity, volatility, fee tier
- Output Panel: Slippage, fees, market impact, net cost, maker/taker, latency, last update
- Orderbook Preview: Live table of top 10 bids/asks with alternating row colors and bold headers
- Connection Status: Visual indicator (colored dot and text) in the status bar
- Color Coding: Output values change color based on thresholds (e.g., green for low slippage, red for high impact)
- Theme Toggle: Switch between light and dark modes with automatic UI updates
- Interactive Charts: Real-time visualization of market data and metrics
- Splitter Controls: Adjust panel sizes to focus on different parts of the interface
- Tooltips: Explanations for all fields

UI Styling

Custom stylesheet with consistent color palette:

```
self.brand_colors = {
    'primary': '#6f42c1',
    'secondary': '#3a1c71',
    'accent': '#8a67dd',
    'success': '#2ecc71',
    'warning': '#f39c12',
    'danger': '#e74c3c',
    'neutral_dark': '#333333',
    'neutral_light': '#f8f9fa',
    'text': '#444444',
    'white': '#ffffff',
    'bid': '#2a9d8f',
    'ask': '#e76f51'
}
```

Dark Mode Implementation

- Theme System with a color dictionary
- Dynamic stylesheet generation
- Signal-Based Updates to propagate theme changes
- Component and Chart Theming
- Persistent Settings across sessions

Interactive Charts

- Built using PyQtGraph
- Price History, Market Depth, Performance Metrics
- Circular buffer for time-series data
- Zoom/pan, hover data inspection, auto-scaling
- Fully integrated with theme changes

Core Models

Almgren-Chriss Market Impact Model

- Estimates cost of large order execution
- Uses risk aversion, volatility, temporary/permanent impact parameters
- Based on optimal execution theory (Almgren & Chriss, 2000)
- Implemented in `src/models/market_impact.py`

Slippage Regression Model

- Predicts expected slippage using linear regression
- Features: order size, depth, volatility, time of day
- Incremental training with new data
- Implemented in `src/models/slippage.py`

Maker/Taker Proportion Model

- Predicts fill probability (maker vs taker)
- Uses logistic regression with features like spread, volatility, size, depth
- Trained incrementally, handles single-class robustness
- Implemented in `src/models/maker_taker.py`

Performance Analysis & Optimization

Latency Benchmarking

- Measures latency from WebSocket reception to UI rendering
- Internal timers for each stage, displayed in UI and logs

Optimization Techniques

- Capped historical buffers and numpy-based arrays
- Asynchronous WebSocket client with reconnection support

- Efficient data structures and thread-safe model computation
- Incremental learning for models
- PyQtGraph for efficient rendering, selective chart updates

Error Handling & Robustness

- WebSocket: Reconnection, thread-safe processing
- Models: Silent error logging, resilience to data issues
- UI: Update-in-progress flag, full exception logging, stable during theme switch

---

Usage Guide

1. `pip install -r requirements.txt`
2. Run using `python main.py`
3. Set parameters, toggle theme, view analytics
4. Monitor `logs/trade_simulator.log`
5. Run tests using `pytest -v`

Extensibility

- Support for new exchanges, models, export features
- Add charts and theme customization
- Configuration persistence for user settings

References

- Almgren, R., & Chriss, N. (2000). *Journal of Risk*
- Binance API Docs: https://binance-docs.github.io/
- OKX API Docs: https://www.okx.com/docs-v5/en/
- scikit-learn: https://scikit-learn.org/
- PyQt6: https://www.riverbankcomputing.com/software/pyqt/
- PyQtGraph: https://www.pyqtgraph.org/