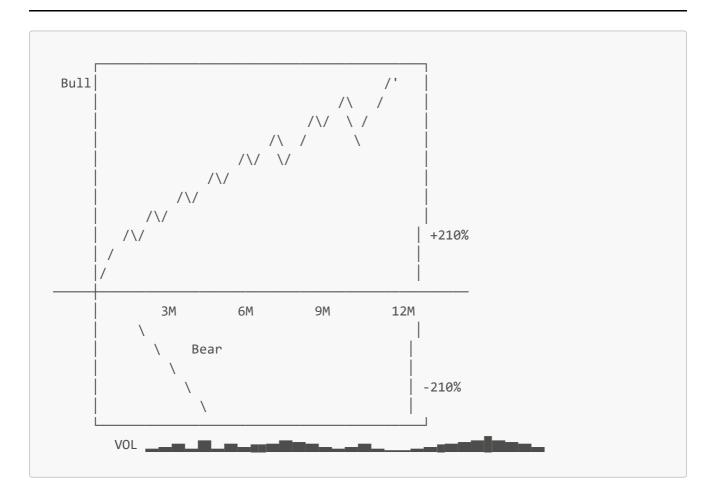
Know-Defeat



Algorithmic Trading System

Overview

A comprehensive algorithmic trading system with sub-minute trading capabilities, probability-based decision making, and dynamic weight adjustments. The system includes tick-level data processing, backtesting capabilities, and real-time market analysis.

Key Features

- Probability engine for trade execution decisions
- Dynamic weight adjustment system for algorithm ranking
- Tick-level data processing and storage
- Real-time market data integration
- Automated backtesting framework
- Performance metrics and analytics

Technical Architecture

- Database: TimescaleDB (PostgreSQL extension) for time-series data
- API Integration: Support for Interactive Brokers, Polygon.io

- Processing: GPU-accelerated calculations for weight optimization
- Storage: Optimized for high-frequency tick data (~350GB for 30 days)

Core Components

1. Probability Engine

- Analyzes historical performance
- Generates success probability metrics
- o Dynamic algorithm ranking

2. Bot Management

- Algorithm combinations
- o Independent operation
- o Ranking-based fund allocation

3. Data Processing

- Tick-level data handling
- Market data compression
- Real-time analytics

Project Structure

- src/
 - o collector/
 - o weight_calculator/
 - o database/
 - validations/
 - resolution/
 - o config/
 - o training/
 - o monitoring/
- database_schema/

Database Schema

```
-- Core tables for tick data and trades

CREATE TABLE tick_data (
    timestamp TIMESTAMP NOT NULL,
    symbol VARCHAR(10) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    volume INTEGER NOT NULL,
    PRIMARY KEY (timestamp, symbol)
);

-- Additional tables for simulated and real trades
```

```
CREATE TABLE simulated_trades (...);
CREATE TABLE real_trades (...);
```

Installation

- 1. Install PostgreSQL and TimescaleDB
- 2. Set up the database:

```
psql -U username postgres
CREATE DATABASE stockdata;
\c stockdata
CREATE EXTENSION IF NOT EXISTS timescaledb;
```

Configuration

- Database path: "C:/Users/[username]/postgres_data"
- Required storage: Minimum 350GB for 30 days of tick data
- Recommended hardware: 32GB RAM, 8+ core CPU
- GPU support: NVIDIA cards recommended for weight calculations

Usage

- 1. Initialize the database
- 2. Configure data sources
- 3. Start the probability engine
- 4. Monitor bot performance through the ranking system

Performance Metrics

- Win rates across multiple timeframes
- Profit per second
- Algorithm rankings
- Risk-adjusted returns

Future Development

- Machine learning integration for weight optimization
- Enhanced circuit breakers
- Bloomberg Terminal integration
- Extended backtesting capabilities

License

[License details to be added]

New Features

- Enhanced bot management with dynamic activation based on rankings.
- Improved backtesting framework with detailed performance metrics.
- · Real-time data processing with optimized tick data handling.

Database Table Descriptions

- tick_data: Stores tick-level market data with timestamps, prices, and volumes.
- **simulated_trades**: Records trades executed during simulations for performance analysis.
- real_trades: Logs actual trades executed in the market for historical tracking.

Instructions for Making Changes

- Code Modularity: Ensure changes are isolated to specific modules to avoid cross-dependencies.
- **Testing**: Run unit tests after making changes to verify functionality.
- Version Control: Use Git for tracking changes and revert if necessary.

Bot Information

- **CoinMomentumBot**: Implements a momentum strategy for COIN with trailing stops.
- Bot Management: Bots are ranked and activated based on performance metrics.
- Strategy Updates: Strategies can be updated dynamically without downtime.

Running the Application

- 1. Start the Database: Ensure your PostgreSQL and TimescaleDB are running.
- 2. **Configure Environment**: Set up your environment variables as needed.
- 3. **Run the Main Script**: Execute the main script to start the application.

python src/main.py

Viewing Logs

- Log Files: Logs are stored in the logs/ directory.
- Real-Time Logs: Use the following command to view logs in real-time:

tail -f logs/application.log

• Log Levels: Adjust log levels in the configuration file to control verbosity.