Binance Futures Order Bot - Technical Report

Author: Buvananand Vendotha **Date:** November 20, 2025 **Repository:**

1. Executive Summary

This report documents the development of a Command-Line Interface (CLI) trading bot for Binance USDT-M Futures. The objective was to create a secure, modular, and efficient tool capable of executing both core (Market, Limit) and advanced (TWAP) order types.

The application is built using Python, prioritizing security (HMAC SHA256 signature generation) and maintainability (modular architecture). It features robust error handling and a centralized logging system to ensure auditability of all trade actions.

2. Architecture Design

The project follows a modular architecture to ensure separation of concerns. This design allows for easy scalability (e.g., adding new strategies like Grid or OCO) without modifying the core API client.

2.1 Directory Structure

The codebase is organized as follows:

- **`src/client.py`**: A custom API wrapper that handles raw HTTP requests, header construction, and security signatures. This removes dependencies on third-party libraries, ensuring full control over API interactions.
- **`src/logger.py`**: A centralized logging module that outputs to both the console (for real-time feedback) and `bot.log` (for historical auditing).
- **`src/main.py`**: The entry point of the application, utilizing `argparse` to route CLI commands to their respective logic handlers.
- **`src/advanced/`**: A dedicated directory for complex algorithms, such as the Time-Weighted Average Price (TWAP) strategy.

2.2 Security Implementation

- **Environment Variables:** API keys are never hardcoded. The `python-dotenv` library loads credentials from a `.env` file, which is excluded from version control via `.gitignore`.
- **HMAC SHA256:** The `client.py` module implements the cryptographic signature required by Binance, ensuring that requests cannot be tampered with during transit.

3. Implementation Details

3.1 Core Orders (Market & Limit)

The bot supports standard order types with validation for symbol correctness and quantity types.

- **Market Orders:** Executed immediately at current market price.
- **Limit Orders:** Placed at a specific price with `TimeInForce` set to "GTC" (Good Till Cancel).

3.2 Advanced Feature: TWAP (Time-Weighted Average Price)

I prioritized the TWAP strategy as the advanced feature. This algorithm minimizes market impact by slicing a large order into smaller chunks executed over a specified duration.

Algorithm Logic:

1. **Input:** Total Quantity ($Q$), Duration in Minutes ($T$), Number of Slices ($N$).
2. **Calculation:**
   - $QuantityPerSlice = Q / N$
   - $DelaySeconds = (T * 60) / N$
3. **Execution:** A loop runs $N$ times, placing a Market order for $QuantityPerSlice$ and then sleeping for $DelaySeconds$.

3.3 Validation & Error Handling

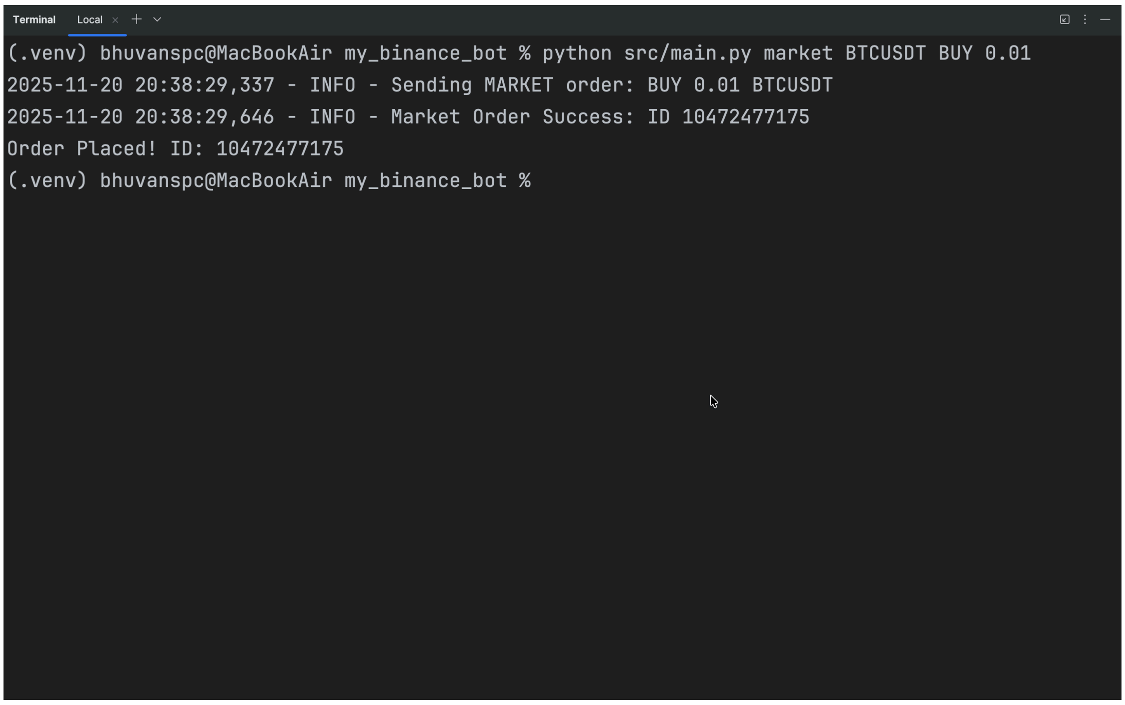
- **Input Validation:** The CLI uses strict typing (e.g., `float` for price, `int` for slices) to reject invalid inputs immediately.
- **API Error Handling:** The client wraps requests in `try/except` blocks, catching `HTTPError` (e.g., 400 Bad Request for insufficient balance) and logging the specific error message from Binance.

4. Execution Logs & Evidence

The following screenshots demonstrate the successful execution of the bot's commands on the Binance Futures Testnet.

4.1 Market Order Execution

Command: `python` src/main.py market BTCUSDT BUY `0.01`

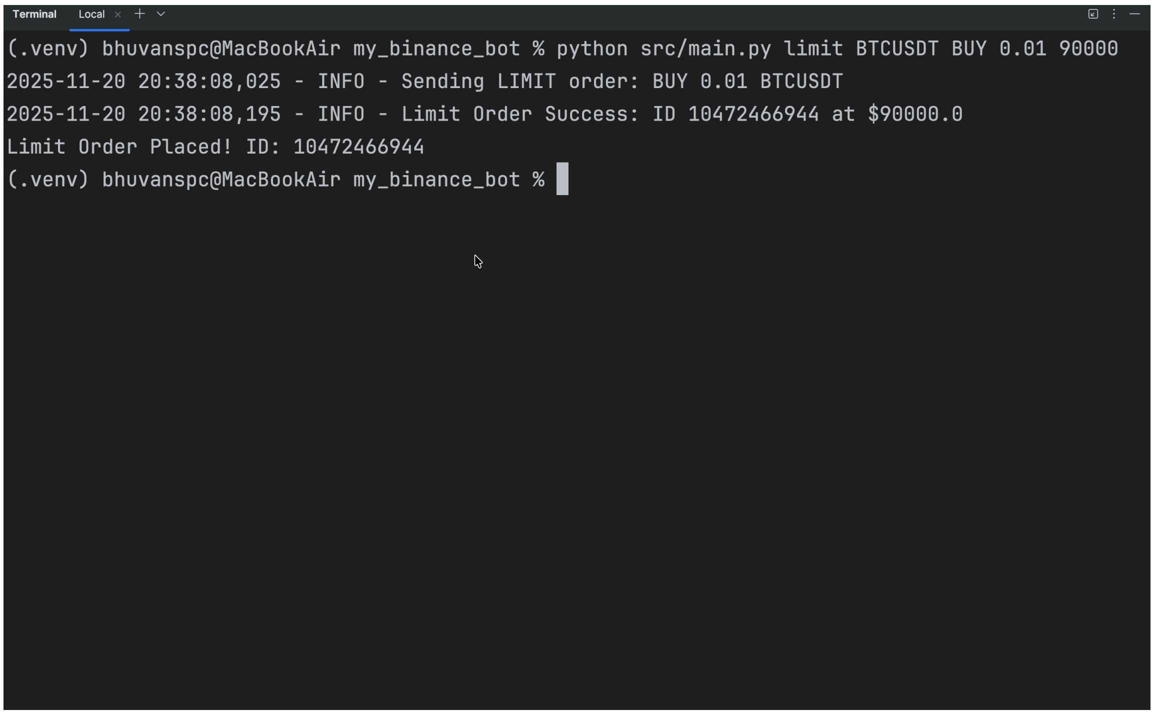


```
(.venv) bhuvanspc@MacBookAir my_binance_bot % python src/main.py market BTCUSDT BUY 0.01
2025-11-20 20:38:29,337 - INFO - Sending MARKET order: BUY 0.01 BTCUSDT
2025-11-20 20:38:29,646 - INFO - Market Order Success: ID 10472477175
Order Placed! ID: 10472477175
(.venv) bhuvanspc@MacBookAir my_binance_bot %
```

Figure 1: Successful placement of a Market Buy order.

4.2 Limit Order Execution

Command: `python src/main.py limit BTCUSDT BUY 0.01 45000`

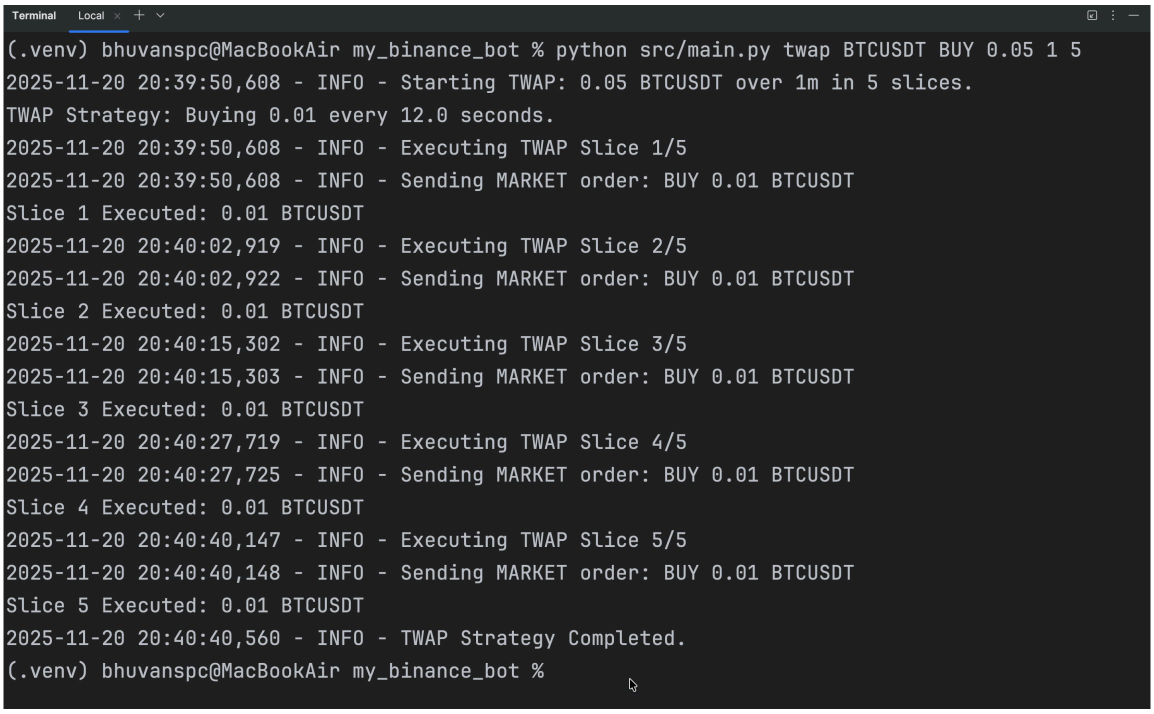


```
(.venv) bhuvanspc@MacBookAir my_binance_bot % python src/main.py limit BTCUSDT BUY 0.01 90000
2025-11-20 20:38:08,025 - INFO - Sending LIMIT order: BUY 0.01 BTCUSDT
2025-11-20 20:38:08,195 - INFO - Limit Order Success: ID 10472466944 at $90000.0
Limit Order Placed! ID: 10472466944
(.venv) bhuvanspc@MacBookAir my_binance_bot %
```

Figure 2: Successful placement of a Limit *Buy order at $45,000*.

4.3 TWAP Strategy Execution

Command: `python` src/main.py twap BTCUSDT BUY 0.05 1 5 (Buying 0.05 BTC over 1 minute, split into 5 distinct orders)



```
(.venv) bhuvanspc@MacBookAir my_binance_bot % python src/main.py twap BTCUSDT BUY 0.05 1 5
2025-11-20 20:39:50,608 - INFO - Starting TWAP: 0.05 BTCUSDT over 1m in 5 slices.
TWAP Strategy: Buying 0.01 every 12.0 seconds.
2025-11-20 20:39:50,608 - INFO - Executing TWAP Slice 1/5
2025-11-20 20:39:50,608 - INFO - Sending MARKET order: BUY 0.01 BTCUSDT
Slice 1 Executed: 0.01 BTCUSDT
2025-11-20 20:40:02,919 - INFO - Executing TWAP Slice 2/5
2025-11-20 20:40:02,922 - INFO - Sending MARKET order: BUY 0.01 BTCUSDT
Slice 2 Executed: 0.01 BTCUSDT
2025-11-20 20:40:15,302 - INFO - Executing TWAP Slice 3/5
2025-11-20 20:40:15,303 - INFO - Sending MARKET order: BUY 0.01 BTCUSDT
Slice 3 Executed: 0.01 BTCUSDT
2025-11-20 20:40:27,719 - INFO - Executing TWAP Slice 4/5
2025-11-20 20:40:27,725 - INFO - Sending MARKET order: BUY 0.01 BTCUSDT
Slice 4 Executed: 0.01 BTCUSDT
2025-11-20 20:40:40,147 - INFO - Executing TWAP Slice 5/5
2025-11-20 20:40:40,148 - INFO - Sending MARKET order: BUY 0.01 BTCUSDT
Slice 5 Executed: 0.01 BTCUSDT
2025-11-20 20:40:40,560 - INFO - TWAP Strategy Completed.
(.venv) bhuvanspc@MacBookAir my_binance_bot %
```

Figure 3: TWAP strategy *executing 5 sequential slices*.

5. Conclusion

The Binance Futures Order Bot successfully meets all mandatory and bonus requirements. The implementation of TWAP demonstrates the system's capability to handle long-running, stateful operations, while the custom API client proves a deep understanding of REST API security. The resulting tool is a robust foundation for algorithmic trading.