User Guide

Getting Started

Installation and Setup

Prerequisites

bash # Required software - C++17 compiler (GCC 7+ or Clang 5+) - CMake 3.10 or higher - Python 3.7+ with pandas, matplotlib, seaborn - Git for version control

1. Optional but recommended

- Visual Studio Code with C++ extensions
- Python virtual environment ```

2. Create build directory

mkdir build && cd build

3. Configure and build

cmake .. make -

4. Verify installation

./tradepulse –help

Download sample data (optional)

python scripts/fetch_binance_ohlcv.py

System Usage Tutorial

1. Running Your First Backtest

Basic Backtest Execution

bash # Start the application ./tradepulse

Select mode: 1 (Backtest with Full Analytics)

Choose assets: 1,2,3 (BTC, ETH, SOL)

Select strategies: 1,2 (EMA-RSI, Momentum)

Expected Output: TradePulse - COMPREHENSIVE Trading System v2.0
==

Select mode: 1. Backtest with Full Analytics 2. Live Shadow with Risk Management 3. Strategy Optimization & Walk-Forward 4. Risk Management Demo Enter choice: 1

Available assets: 1. BTCUSDT 2. ETHUSDT 3. SOLUSDT Select assets (comma-separated indices): 1,2

Available strategies: 1. EMARSI 2. Momentum 3. SMA 4. Volatility 5. Breakout Select strategies (comma-separated indices): 1,2

Understanding Results

After backtesting completes, you'll see:

[EMARSI - BTCUSDT] Results: - Total Trades: 45 - Win Rate: 62.2% - Cumulative Return: 15.7% - Sharpe Ratio: 1.23 - Max Drawdown: 8.4% - Avg Slippage: 2.1 bps ```

2. Live Shadow Trading

Setting Up Live Data Streams

bash # Terminal 1: Start BTC data stream python scripts/binance_stream.py 1

Terminal 2: Start ETH data stream

python scripts/binance_stream.py 2

Terminal 3: Run live shadow system

./tradepulse # Select mode: 2 (Live Shadow with Risk Management)

Monitoring Live Performance

The live system provides real-time updates:

BTCUSDT @ 2024-01-15 10:30:00 | Price: $42,150 | Vol: 1.25 | Regime: ⍰ BULL BUY APPROVED: EMARSI | BTCUSDT | $42,150 | Size: 8% ⍰ Portfolio: $108,450.00 | P&L: $8,450.00 | Positions: 2

3. Strategy Development

Creating a New Strategy

1. **Define Strategy Class:** ```cpp // include/strategies/strategy_custom.hpp #pragma once #include "strategy.hpp"

class CustomStrategy : public Strategy { public: void on_data(const Candle& candle, const Candle* high_tf = nullptr) override; bool should_buy() const override; bool should_sell() const override; void enable_debug(const std::string& strategy_name, const std::string& symbol) override;

private: // Strategy-specific variables bool buy_signal = false; bool sell_signal = false; // Add your indicators and parameters }; ```

2. **Implement Strategy Logic:** ```cpp // src/strategies/strategy_custom.cpp #include "strategies/strategy_custom.hpp"

void CustomStrategy::on_data(const Candle& candle, const Candle* high_tf) { // Reset signals buy_signal = sell_signal = false;

```
// Your strategy logic here
if (/* your buy condition */) {
    buy_signal = true;
}

if (/* your sell condition */) {
    sell_signal = true;
}
```

}

bool CustomStrategy::should_buy() const { return buy_signal; } bool CustomStrategy::should_sell() const { return sell_signal; } ```

3. **Register Strategy:** ```cpp // In main.cpp, add to strategy list: {"Custom", { return std::make_unique(); }} ```

Strategy Parameters

Common Parameters to Consider: - **Lookback Periods**: How many candles to analyze - **Thresholds**: Signal generation thresholds - **Risk Controls**: Stop loss, take profit levels - **Filters**: Volume, volatility, trend filters

Example Parameter Configuration: ```cpp class MomentumStrategy : public Strategy { private: int momentum_period = 10; // Lookback period double threshold_pct = 0.5; // Signal threshold double volume_multiplier = 1.2; // Volume filter int cooldown_candles = 5; // Prevent overtrading };

4. Performance Analysis

Generated Reports

After backtesting, the system generates several outputs:

1. Trade Logs (`logs/` directory): - `trades_STRATEGY_SYMBOL.csv`: Individual trade records - `STRATEGY_SYMBOL_equity.csv`: Portfolio value over time - `debug_STRATEGY_SYMBOL.csv`: Detailed strategy signals

2. Performance Charts (`plots/` directory): - `STRATEGY_equity_curve.png`: Portfolio growth over time - `STRATEGY_drawdown.png`: Drawdown analysis - `STRATEGY_win_loss_distribution.png`: Trade outcome distribution - `STRATEGY_pnl_histogram.png`: Profit/loss distribution

3. HTML Reports (`reports/` directory): - Comprehensive performance analysis - Risk metrics and statistics - Interactive charts and tables

Interpreting Results

Key Metrics to Monitor:

1. **Sharpe Ratio**: Risk-adjusted returns
   - 1.0: Good performance
   - 2.0: Excellent performance
   - < 0.5: Poor risk-adjusted returns
2. **Maximum Drawdown**: Largest peak-to-trough decline
   - < 10%: Conservative strategy
   - 10-20%: Moderate risk
   - 20%: High risk strategy
3. **Win Rate**: Percentage of profitable trades
   - 60%: High accuracy strategy
   - 40-60%: Balanced approach
   - < 40%: Requires high profit factor
4. **Profit Factor**: Gross profit / Gross loss

  - 2.0: Strong strategy
  - 1.5-2.0: Acceptable
  - < 1.2: Marginal profitability

5. Strategy Optimization

Parameter Optimization

bash # Run optimization mode ./tradepulse # Select mode: 3 (Strategy Optimization & Walk-Forward)

The system will: 1. Test multiple parameter combinations 2. Perform walk-forward analysis 3. Generate stability and robustness scores 4. Save results to `reports/optimization_results.txt`

Walk-Forward Analysis

Process: 1. **In-Sample Period**: Optimize parameters on historical data 2. **Out-of-Sample Period**: Test optimized parameters on unseen data 3. **Rolling Window**: Repeat process with advancing time windows 4. **Stability Analysis**: Compare in-sample vs out-of-sample performance

Interpreting Results: - **Stability Score**: Consistency of out-of-sample performance - **Robustness Score**: Strategy's ability to maintain performance - **Parameter Sensitivity**: How sensitive strategy is to parameter changes

6. Risk Management

Risk Limits Configuration

Default Risk Limits:

cpp RiskLimits limits; limits.max_position_size = 0.1; // 10% max position limits.max_daily_loss = 0.02; // 2% daily loss limit limits.max_drawdown = 0.05; // 5% max drawdown limits.max_positions = 5; // Max concurrent positions

Customizing Risk Limits:

cpp // Modify in main.cpp or create configuration file RiskLimits custom_limits; custom_limits.max_position_size = 0.15; // 15% max position custom_limits.max_daily_loss = 0.03; // 3% daily loss limit custom_limits.max_drawdown = 0.08; // 8% max drawdown

Risk Monitoring

Real-Time Alerts: - Position size violations - Correlation limit breaches - Drawdown threshold approaches - Daily loss limit warnings

Risk Metrics Dashboard: Risk Alerts: Approaching maximum drawdown limit High correlation detected between BTCUSDT and ETHUSDT All other risk metrics within limits

Troubleshooting and FAQ

Common Issues

1. Build Errors

Problem: CMake configuration fails bash CMake Error: Could not find CMAKE_CXX_COMPILER

Solution: bash # Install build tools sudo apt-get install build-essential cmake # Ubuntu/Debian brew install cmake # macOS

Problem: Linking errors with filesystem bash undefined reference to
std::filesystem::create_directories' \``

Solution: The CMakeLists.txt handles this automatically, but if issues persist: ```bash # For older GCC versions export CXXFLAGS="-lstdc++fs"

2. Data Issues

Problem: No data found for backtesting

bash No data found for optimization

Solution: bash # Check data directory structure ls -la data/ # Should contain: btc_usdt_1m.csv, eth_usdt_1m.csv, sol_usdt_1m.csv

Download sample data

python scripts/fetch_binance_ohlcv.py

Problem: Live data stream connection fails

bash Connection error: [Errno 111] Connection refused

Solution: bash # Check internet connection ping stream.binance.com

Verify Python dependencies

pip install websockets asyncio

Try different symbol

python scripts/binance_stream.py 2 # ETH instead of BTC

3. Performance Issues

Problem: Slow backtesting performance

bash Processing takes too long for large datasets

Solution: bash # Reduce dataset size for testing head -10000 data/btc_usdt_1m.csv > data/btc_test.csv

Use fewer strategies simultaneously

Optimize compiler flags

cmake -DCMAKE_BUILD_TYPE=Release ..

4. Memory Issues

Problem: Out of memory errors bash std::bad_alloc

Solution: bash # Monitor memory usage top -p $(pgrep tradepulse)

Reduce lookback periods in strategies

Process smaller date ranges

Increase system swap space

Frequently Asked Questions

Q: How do I add a new data source?

A: Modify `src/data_loader.cpp` to support your format: ```cpp // Add new parsing logic for your data format std::vector load_custom_data(const std::string& filename) { // Your implementation here } ```

Q: Can I run multiple strategies on the same asset?

A: Yes, the system supports multiple strategies per asset. Each strategy runs independently with its own trade engine.

Q: How do I modify slippage and latency settings?

A: Edit the parameters in `main.cpp`: cpp double slippage_bps = 2.0; // 2 basis points int latency_sec = 5; // 5 seconds delay

Q: How do I export results to Excel?

A: The system generates CSV files that can be opened in Excel: - Trade logs: `logs/trades_*.csv` - Equity curves: `logs/*_equity.csv` - System metrics: `reports/system_metrics.csv`

Q: Can I run the system on Windows?

A: Yes, the code is cross-platform. Use Visual Studio 2019+ or MinGW-w64 for compilation.

Q: How do I optimize strategy parameters?

A: Use the built-in optimization mode (option 3) or modify parameters directly in strategy constructors and recompile.

Q: What's the difference between backtest and live shadow modes?

A: - **Backtest**: Uses historical data, processes all at once - **Live Shadow**: Uses real-time data, simulates trading without actual orders

Q: How do I interpret the Sharpe ratio?

A: - < 0: Strategy loses money - 0-1: Positive returns but high volatility - 1-2: Good risk-adjusted returns - > 2: Excellent risk-adjusted returns

Q: Can I use this for live trading?

A: This system is designed for backtesting and paper trading only. For live trading, additional infrastructure for order routing and execution would be needed.

Getting Help

If you encounter issues not covered in this guide:

1. **Check Log Files**: Look in `logs/` directory for error messages
2. **Enable Debug Mode**: Strategies have debug logging capabilities
3. **System Monitoring**: Use the system monitor for performance insights
4. **Code Documentation**: Review inline comments in source files

Performance Tips

1. **Data Management**: Keep only necessary historical data
2. **Strategy Optimization**: Avoid complex calculations in hot paths
3. **Memory Usage**: Monitor memory consumption with large datasets
4. **Parallel Processing**: The system supports multi-threading for multiple strategies
5. **Compiler Optimization**: Use Release build for production runs