

IntelliSense User Guide

Complete Usage Manual for Your Trading Intelligence Platform

Table of Contents

1. [Getting Started](#)
 2. [User Interface Options](#)
 3. [Data Collection Methods](#)
 4. [Running IntelliSense Sessions](#)
 5. [Parallel Operations Guide](#)
 6. [Daily Workflow Examples](#)
 7. [GUI Interface Guide](#)
 8. [Command Line Interface](#)
 9. [Configuration Management](#)
 10. [Troubleshooting Guide](#)
 11. [Best Practices](#)
 12. [Advanced Usage Scenarios](#)
-

Getting Started

What You Need to Know

IntelliSense can operate in multiple ways to fit your trading workflow:

1. Background Data Collection Mode

- Runs alongside your normal TESTRADE usage
- Zero impact on your trading performance
- Automatically collects correlation data for later analysis
- No GUI needed - works silently in background

2. Dedicated Analysis Mode

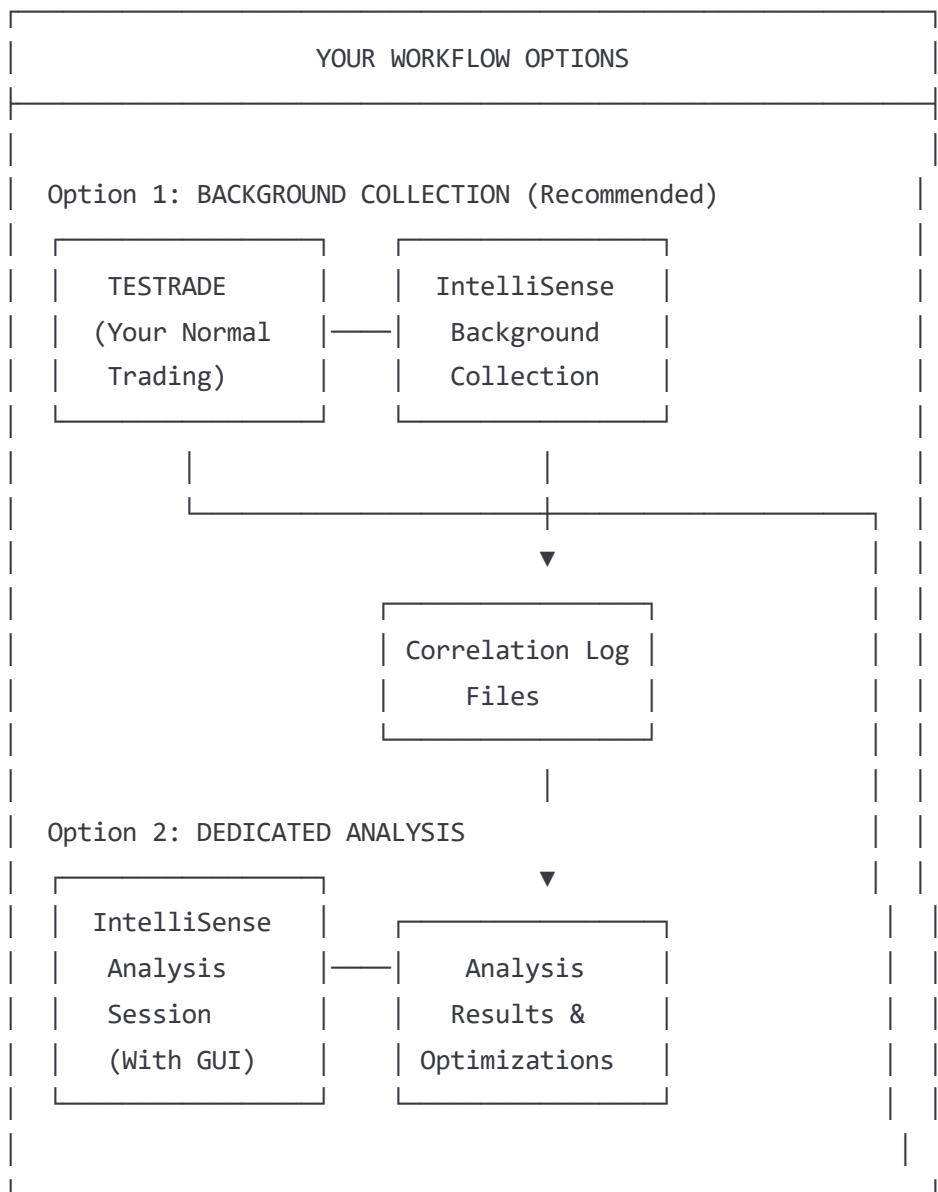
- Separate instance for replay and analysis
- Full GUI interface for interactive exploration

- Uses previously collected correlation data
- Can run on same machine or different machine

3. Controlled Injection Mode

- Dedicated session for safe experimentation
- Advanced GUI for experiment design and monitoring
- Isolated from your live trading
- Requires careful setup and oversight

System Architecture Overview



User Interface Options

1. GUI Interface (Recommended for Analysis)

IntelliSense Dashboard

```
python

# GUI Application Entry Point
python -m intellisense.gui.main_dashboard
```

Features:

- **Session Management:** Create, load, and manage analysis sessions
- **Real-Time Monitoring:** Live performance metrics during data collection
- **Interactive Analysis:** Visual exploration of optimization opportunities
- **Experiment Designer:** GUI for setting up controlled injection experiments
- **Results Visualization:** Charts, graphs, and performance comparisons

GUI Components

Main Dashboard


IntelliSense Trading Intelligence Platform


COLLECT DATA


ANALYZE SESSION

EXPERIMENT SAFELY

Active Sessions:

 Background Collection: ACTIVE (2h 34m)

 Analysis Session: session_20241205_morning

 Last Experiment: controlled_injection_test_1

Quick Actions:

[Start Collection]

[New Analysis]

[View Results]

Data Collection Panel

Data Collection Status		
OCR	PRICE	BROKER
<div><div></div>ACTIVE</div>	<div><div></div>ACTIVE</div>	<div><div></div>ACTIVE</div>
1,247 evts	8,392 evts	83 evts
Session: background_collection_20241205		
Duration: 2h 34m 17s		
Data Quality: Excellent (99.7% correlated events)		
<div>[Pause Collection] [Stop & Analyze] [Settings]</div>		

Analysis Results Panel

Analysis Results - Session: morning_optimization	
Performance Improvements Found:	
<div>OCR OPTIMIZATION</div>	
Current Avg Latency: 15.3ms	
Optimized Latency: 12.1ms (3.2ms improvement)	
Confidence: 94% Impact: \$1,247/day	
Recommendation: Increase OCR threads from 2 to 4	
<div>[Apply Optimization] [Test Safely] [More Details]</div>	
<div>BROKER OPTIMIZATION</div>	
Current Avg Response: 8.7ms	
Potential Improvement: 1.4ms (order timeout adjustment)	
Confidence: 76% Impact: \$423/day	
Recommendation: Adjust order timeout from 50ms to 35ms	
<div>[Apply Optimization] [Test Safely] [More Details]</div>	

2. Command Line Interface (For Automation)

CLI Commands

```
bash
```

```
# Start background data collection
```

```
intellisense collect start --session-name "morning_session"
```

```
# Run analysis on collected data
```

```
intellisense analyze --session-path "./sessions/morning_session" --output-format gui
```

```
# Apply optimization safely
```

```
intellisense optimize apply --recommendation-id "ocr_threads_001" --validation-mode safe
```

```
# Run controlled experiment
```

```
intellisense experiment run --config "./experiments/signal_timing_test.yaml"
```

3. Web Interface (Future Enhancement)

- **Browser-based dashboard** for remote monitoring
 - **REST API** for integration with other tools
 - **Webhook notifications** for optimization alerts
 - **Mobile-responsive** design for monitoring on-the-go
-

Data Collection Methods

Method 1: Background Collection (Recommended)

How It Works

```
python
```

```
# IntelliSense runs as background service alongside TESTRADE
```

```
class BackgroundCollectionService:
```

```
    def start_background_collection(self):
```

```
        # Activates enhanced components in your existing TESTRADE instance
```

```
        self.activate_data_capture_mode()
```

```
        # Starts collecting correlation data silently
```

```
        self.start_correlation_logging()
```

```
        # Zero impact on your trading performance
```

```
        # Data saved to: C:/TESTRADE/intellisense_sessions/
```

Setup Process

1. One-Time Configuration

```
bash

# Configure IntelliSense for background collection
intellisense config setup --mode background

# Test the configuration
intellisense config test --verify-integration
```

2. Start Collection

```
bash

# Start collecting data (runs until you stop it)
intellisense collect start --session-name "daily_collection"
```

3. Your Normal Trading

- Trade normally with TESTRADE
- Zero performance impact
- Data automatically collected in background
- Small log files created with timing data

4. Stop Collection

```
bash

# Stop when you want to analyze
intellisense collect stop --analyze-now
```

What Gets Collected

yaml

Data Collected Automatically:

OCR Events:

- Frame processing timestamps
- OCR result data
- Processing latency measurements
- Confidence scores

Price Events:

- Market data reception timestamps
- Price tick processing latency
- Data source information
- Quote/trade classifications

Broker Events:

- Order acknowledgment timestamps
- Fill confirmation timing
- Response processing latency
- Order status changes

File Locations:

Session Directory: "C:/TESTRADE/intellisense_sessions/{session_name}/"

OCR Data: "ocr_correlation.jsonl"

Price Data: "price_correlation.jsonl"

Broker Data: "broker_correlation.jsonl"

Session Config: "session_config.json"

Method 2: Dedicated Collection Session

When to Use

- Testing specific scenarios
- Collecting data for particular market conditions
- Running controlled experiments
- Isolating specific trading strategies

Setup Process

```
python
```

```
# Create dedicated collection session
```

```
intellisense session create --name "volatility_test" --mode dedicated
```

```
# Configure specific collection parameters
```

```
intellisense session config --symbols "AAPL,MSFT,GOOGL" --duration "2h"
```

```
# Start dedicated collection
```

```
intellisense session start --with-gui
```

Method 3: Controlled Injection Collection

Advanced Data Collection with Safe Trading

```
python
```

```
# ADVANCED: Controlled injection for optimization testing
```

```
intellisense experiment create --name "signal_timing_optimization"
```

```
# Configure safe test trades
```

```
intellisense experiment config \
```

```
  --symbols "AAPL" \
```

```
  --max-position 10 \
```

```
  --test-account "paper_trading" \
```

```
  --isolation-mode "full"
```

```
# Execute controlled experiment
```

```
intellisense experiment run --with-monitoring-gui
```

Safety Features

- **Position Isolation:** Test trades don't affect your real positions
- **Separate Account:** Uses paper trading or separate broker account
- **Automatic Limits:** Maximum position sizes and exposure limits
- **Emergency Stop:** Immediate halt and cleanup if needed

Running IntelliSense Sessions

Session Types Explained

1. Background Collection Session

Purpose: Collect data while trading normally **Duration:** Hours to days **Impact:** Zero performance impact
Output: Correlation logs for later analysis

```
bash

# Start background collection
intellisense collect start --session-name "week_1_data"

# Check status anytime
intellisense collect status

# Stop when ready to analyze
intellisense collect stop
```

2. Analysis Session

Purpose: Analyze collected data and find optimizations **Duration:** Minutes to hours **Impact:** No impact on live trading **Output:** Optimization recommendations and performance insights

```
bash

# Run analysis on collected data
intellisense analyze start \
  --session-path "./sessions/week_1_data" \
  --gui \
  --engines "ocr,price,broker"

# View results in GUI or generate report
intellisense analyze report --format html
```

3. Optimization Testing Session

Purpose: Safely test optimization recommendations **Duration:** Minutes to hours **Impact:** No impact on live trading (uses simulation) **Output:** Validated optimizations ready for deployment

```
bash

# Test optimization safely before applying
intellisense optimize test \
  --recommendation-id "ocr_threads_001" \
  --simulation-mode \
  --confidence-threshold 90
```

4. Controlled Injection Session

Purpose: Generate controlled data for optimization research **Duration:** Minutes to hours **Impact:** Controlled test trades (isolated from live trading) **Output:** High-precision optimization data

```
bash

# Advanced: Controlled injection experiment
intellisense experiment run \
  --config "./experiments/latency_optimization.yaml" \
  --safety-mode strict \
  --gui
```

Session Management

Creating Sessions

```
python

# Python API for session management
from intellisense import SessionManager

session_manager = SessionManager()

# Create different types of sessions
background_session = session_manager.create_background_session(
    name="daily_collection",
    duration_hours=8,
    symbols=["AAPL", "MSFT", "GOOGL"]
)

analysis_session = session_manager.create_analysis_session(
    name="optimization_analysis",
    data_source="./sessions/daily_collection",
    engines=["ocr", "price", "broker"]
)
```

Session Configuration Files

yaml

Example: background_collection_config.yaml

session_config:

name: "daily_background_collection"

type: "background_collection"

duration: "8h"

data_collection:

ocr_events: true

price_events: true

broker_events: true

correlation_logging: true

performance:

max_latency_overhead_us: 100

queue_size: 10000

flush_interval_ms: 1000

output:

base_path: "C:/TESTRADE/intellisense_sessions"

compression: true

encryption: false

yaml

Example: analysis_session_config.yaml

analysis_config:

name: "morning_optimization_analysis"

type: "analysis"

input:

session_path: "./sessions/daily_background_collection"

engines:

ocr_intelligence:

enabled: true

validation_threshold: 0.95

performance_analysis: true

broker_intelligence:

enabled: true

latency_analysis: true

order_validation: true

price_intelligence:

enabled: true

feed_analysis: true

timing_analysis: true

output:

results_format: ["json", "html", "gui"]

recommendations_file: "optimization_recommendations.json"

detailed_analysis: true

Parallel Operations Guide

Can I Run IntelliSense While Trading?

✅ YES - Background Collection Mode (Recommended)

This is the designed workflow:

Your Normal Day:

9:30 AM: Start background collection	
intellisense collect start --session "today"	
9:30 AM - 4:00 PM: Trade normally with TESTRADE	
- Zero performance impact	
- Data collected automatically	
- No GUI needed	
4:00 PM: Stop collection and analyze	
intellisense collect stop --analyze-now	
4:15 PM: Review optimization recommendations	
intellisense gui analyze --session "today"	
Evening: Apply safe optimizations for tomorrow	
intellisense optimize apply --safe-only	

Multiple Instance Options

Option 1: Same Machine, Background Service

bash

Terminal 1: Your normal TESTRADE

`./testtrade.exe`

Terminal 2: Start IntelliSense background collection

`intellisense collect start --session "live_data" --background`

Terminal 3: (Later) Analyze in separate session

`intellisense analyze --session "./sessions/live_data" --gui`

Option 2: Separate Machines

```
bash
```

```
# Trading Machine: Run TESTRADE + Background Collection
```

```
intellisense collect start --session "trading_data" --sync-to "analysis_machine"
```

```
# Analysis Machine: Receive data and analyze
```

```
intellisense analyze --remote-session "trading_machine:/sessions/trading_data" --gui
```

Option 3: Time-Separated Workflow

```
bash
```

```
# During Trading Hours: Only collect data
```

```
intellisense collect start --session "market_hours" --quiet
```

```
# After Market Close: Analyze collected data
```

```
intellisense collect stop
```

```
intellisense analyze start --session "./sessions/market_hours" --full-analysis
```

Resource Usage

Background Collection Impact

```
yaml
```

CPU Usage: < 2% additional overhead

Memory Usage: < 500MB additional RAM

Disk I/O: < 10MB/hour of correlation logs

Network: 0 additional network usage

Latency Impact:

OCR Processing: < 0.1ms additional latency

Price Processing: < 0.05ms additional latency

Broker Processing: < 0.1ms additional latency

Analysis Session Resources

yaml

CPU Usage: 20-50% during analysis (separate process)

Memory Usage: 2-8GB during analysis (depends on data size)

Disk I/O: Heavy read of correlation logs during analysis

Network: 0 (unless using remote data)

Note: Analysis runs in completely separate process from trading

Daily Workflow Examples


Typical Day 1: Background Collection + Evening Analysis

Morning Setup (2 minutes)

```
bash

# 9:25 AM - Before market open
cd C:/TESTRADE
intellisense collect start --session "$(date +%Y%m%d)_trading" --background

# Verify collection is running
intellisense status

# Output:  Background collection active (0 events collected)
```

Normal Trading (All Day)

- Use TESTRADE exactly as normal
- No difference in performance or behavior
- IntelliSense silently collects timing data
- Small correlation log files created automatically

Optional: Check Status During Day

```
bash

# Quick status check (optional)
intellisense status

# Output:  Background collection active (1,247 OCR, 8,392 Price, 83 Broker events)
```

End of Day Analysis (30 minutes)

```
bash
```

```
# 4:00 PM - Market close
```

```
intellisense collect stop --session "$(date +%Y%m%d)_trading"
```

```
# Start analysis with GUI
```

```
intellisense analyze start --session "./sessions/$(date +%Y%m%d)_trading" --gui
```

```
# Analysis GUI opens showing:
```

```
# - Performance bottlenecks found
```

```
# - Optimization recommendations
```

```
# - Confidence levels
```

```
# - Estimated profit impact
```

Apply Optimizations (15 minutes)

```
bash
```

```
# Apply safe optimizations for tomorrow
```

```
intellisense optimize apply --safe-only --schedule "next_trading_day"
```

```
# Test risky optimizations in simulation
```

```
intellisense optimize test --medium-risk --simulation-mode
```

Typical Day 2: Dedicated Analysis Session

Use Case: Deep Analysis of Specific Trading Session

```
bash
```

```
# Load yesterday's data for detailed analysis
```

```
intellisense session load --path "./sessions/20241204_trading"
```

```
# Start comprehensive analysis with GUI
```

```
intellisense analyze comprehensive --gui --engines all
```

```
# GUI provides:
```

```
# - Detailed latency breakdowns
```

```
# - Frame-by-frame OCR analysis
```

```
# - Trade-by-trade broker response analysis
```

```
# - Market condition correlations
```

```
# - Parameter sensitivity analysis
```


Weekly Optimization Routine

Sunday Evening: Weekly Review

```
bash

# Analyze entire week's trading data
intellisense analyze weekly \
  --sessions "./sessions/2024W49_*" \
  --comparison-mode \
  --generate-report

# Output: weekly_optimization_report.html
# - Week-over-week performance trends
# - Cumulative optimization opportunities
# - Market condition adaptations needed
# - Strategic recommendations
```

Advanced: Controlled Injection Day

Use Case: Testing New Strategy Parameters

```
bash

# Setup controlled injection experiment
intellisense experiment create \
  --name "signal_timeout_optimization" \
  --config "./experiments/signal_timeout_test.yaml"




# Run safe controlled injection with GUI monitoring
intellisense experiment run \
  --name "signal_timeout_optimization" \
  --gui \
  --safety-mode strict

# GUI shows:
# - Real-time experiment progress
# - Safety monitoring dashboards
# - Preliminary results
# - Emergency stop controls
```






GUI Interface Guide

Main Dashboard Components

1. Session Manager

Session Manager			
Recent Sessions:			
<div><div><div></div><div>20241205_trading</div><div>[COLLECTING] [ANALYZE]</div></div><div><div></div><div>20241204_analysis</div><div>[COMPLETED] [VIEW RESULTS]</div></div><div><div></div><div>signal_timing_exp</div><div>[COMPLETED] [VIEW RESULTS]</div></div></div>			
Quick Actions:			
<div>[New Collection Session] [New Analysis] [New Experiment]</div>			
Templates:			
<div>[Daily Collection] [Weekly Analysis] [Parameter Test]</div>			

2. Real-Time Monitoring

Live Performance Monitor	
OCR PERFORMANCE	BROKER PERFORMANCE
Current Latency: 15.3ms	Response Time: 8.7ms
Target: < 12ms	Target: < 10ms
Status:  OPTIMIZATION OPPORTUNITY	Status:  GOOD
Last 100 frames:	Recent Orders:
<div><div></div></div> 87%	 AAPL BUY 100 (7.2ms)
	 MSFT SELL 50 (9.1ms)
PRICE FEED PERFORMANCE	
Feed Latency: 2.1ms	Events/sec: 847
Status:  EXCELLENT	Last Update: 09:34:22.147

3. Analysis Results Viewer

Analysis Results - Session: 20241205_morning		
<div>🎯 OPTIMIZATION OPPORTUNITIES FOUND: 3</div>		
<div><div>HIGH IMPACT</div><div><div>🔧 OCR Thread Optimization</div><div>Current: 15.3ms avg → Optimized: 12.1ms (-3.2ms)</div><div>Confidence: 94% Daily Impact: \$1,247</div><div>[Apply Now] [Test First] [More Details]</div></div></div>		
<div><div>MEDIUM IMPACT</div><div><div>🕒 Signal Timeout Adjustment</div><div>Current: 50ms → Optimized: 35ms (-15ms)</div><div>Confidence: 76% Daily Impact: \$423</div><div>[Apply Now] [Test First] [More Details]</div></div></div>		
<div><div>EXPERIMENTAL</div><div><div>🧪 Advanced OCR Configuration</div><div>Potential improvement: 2.1ms</div><div>Confidence: 52% Requires Testing</div><div>[Design Experiment] [More Details]</div></div></div>		

4. Experiment Designer

Controlled Injection Experiment Designer	
Experiment Name: signal_timeout_optimization_v2	
SAFETY CONFIGURATION	
Isolation Mode:	<input checked="" type="checkbox"/> Full Position Isolation
Max Position Size:	[100] shares per symbol
Max Total Exposure:	[\$10,000]
Auto-Stop Trigger:	<input checked="" type="checkbox"/> 5-minute timeout
Emergency Contact:	[trader@company.com]
EXPERIMENT PARAMETERS	
Parameter:	Signal Timeout
Current Value:	[50]ms
Test Values:	[30, 35, 40, 45]ms
Test Symbols:	[AAPL] [MSFT] [GOOGL]
Test Duration:	[30] minutes per configuration
Sample Size:	[100] trades per test
[Preview Experiment] [Run Safety Check] [Start Experiment]	

GUI Navigation

Main Menu Structure

File

- └─ New Session
 - | └─ Background Collection
 - | └─ Analysis Session
 - | └─ Controlled Experiment
- └─ Open Session
- └─ Recent Sessions
- └─ Export Results

Tools

- └─ Configuration Manager
- └─ Performance Monitor
- └─ Optimization Recommendations
- └─ Safety Checks

View

- └─ Dashboard
- └─ Real-Time Monitoring
- └─ Analysis Results
- └─ Experiment Designer
- └─ System Status

Help

- └─ User Guide
- └─ Tutorial Videos
- └─ Troubleshooting
- └─ Contact Support

Keyboard Shortcuts

yaml

Global Shortcuts:

- Ctrl+N: New Session
- Ctrl+O: Open Session
- Ctrl+S: Save Results
- Ctrl+R: Refresh Data
- F5: Refresh Real-Time Data
- Escape: Emergency Stop (during experiments)

Analysis Mode:

- Space: Play/Pause Replay
- ←/→: Previous/Next Event
- Ctrl+F: Find Specific Event
- Ctrl+Z: Zoom to Selection

Monitoring Mode:

- F1: Toggle OCR Monitor
- F2: Toggle Price Monitor
- F3: Toggle Broker Monitor
- F12: Full Screen Mode

Command Line Interface

Basic Commands

Data Collection

bash

Start background collection

```
intellisense collect start [options]
  --session-name TEXT      Session name
  --duration TEXT          Duration (e.g., "8h", "all_day")
  --symbols TEXT           Comma-separated symbols
  --background             Run as background service
  --quiet                  Minimal output
  --config FILE            Custom configuration file
```

Stop collection

```
intellisense collect stop [options]
  --session-name TEXT      Session to stop
  --analyze-now            Start analysis immediately
  --save-config            Save session config for reuse
```

Check collection status

```
intellisense collect status [options]
  --session-name TEXT      Specific session status
  --all                    All active sessions
  --detailed               Detailed statistics
```

Analysis Commands

bash

Run analysis

```
intellisense analyze start [options]
  --session-path PATH      Path to collected data
  --engines TEXT           Engines to run (ocr,price,broker,all)
  --gui                    Open GUI interface
  --background             Run analysis in background
  --output-format TEXT     Output format (json,html,csv)
```

Generate reports

```
intellisense analyze report [options]
  --session-path PATH      Analysis session path
  --format TEXT            Report format (html,pdf,json)
  --template TEXT          Report template
  --output-file PATH       Output file path
```

Optimization Commands

bash

Apply optimizations

intellisense optimize apply [options]

--recommendation-id TEXT Specific recommendation ID
--safe-only Apply only low-risk optimizations
--test-first Test **in** simulation before applying
--schedule TEXT Schedule **for** deployment (now,market_close,next_day)
--backup-config Backup current config before changes

Test optimizations

intellisense optimize **test** [options]

--recommendation-id TEXT Test specific recommendation
--simulation-mode Test **in** simulation environment
--confidence-threshold INT Minimum confidence level (**0**-100)
--duration TEXT Test duration
--rollback-on-failure Auto-rollback **if test** fails

List recommendations

intellisense optimize list [options]

--session-path **PATH** Session with recommendations
--filter TEXT Filter by impact level (high,medium,low)
--sort-by TEXT Sort by (impact,confidence,risk)

Experiment Commands

bash

Create controlled experiment

intellisense experiment create [options]

--name TEXT	Experiment name
--config FILE	Experiment configuration file
--template TEXT	Use predefined template
--interactive	Interactive experiment setup

Run experiment

intellisense experiment run [options]

--name TEXT	Experiment name
--gui	Run with GUI monitoring
--safety-mode TEXT	Safety level (strict,normal,relaxed)
--dry-run	Simulate without real trades
--emergency-contact TEXT	Emergency contact email

Monitor experiment

intellisense experiment monitor [options]

--name TEXT	Experiment name
--gui	Open monitoring GUI
--alerts	Enable safety alerts
--auto-stop TEXT	Auto-stop conditions

Advanced CLI Usage

Batch Operations

bash

Process multiple sessions

```
intellisense batch analyze \  
  --sessions-dir "./sessions/2024W49/" \  
  --output-dir "./analysis_results/" \  
  --parallel 4 \  
  --email-report "trader@company.com"
```

Automated daily routine

```
intellisense daily-routine \  
  --collect-start "09:25" \  
  --collect-stop "16:05" \  
  --analyze-immediate \  
  --apply-safe-optimizations \  
  --email-summary "manager@company.com"
```

Configuration Management

bash

Manage configurations

intellisense config list	<i># List all configurations</i>
intellisense config show trading	<i># Show specific config</i>
intellisense config backup	<i># Backup current config</i>
intellisense config restore backup_20241205	<i># Restore from backup</i>
intellisense config validate	<i># Validate current config</i>

Integration Commands

bash

Export to external systems

```
intellisense export prometheus \  
  --session-path "./sessions/today" \  
  --metrics-file "./metrics/trading_performance.prom"
```

```
intellisense export grafana \  
  --session-path "./sessions/today" \  
  --dashboard-config "./grafana/intellisense_dashboard.json"
```

Import from external sources

```
intellisense import trading-logs \  
  --log-path "./logs/testtrade_20241205.log" \  
  --format "testtrade_native" \  
  --output-session "./sessions/imported_20241205"
```

CLI Configuration Files

Default CLI Configuration

yaml

```
# ~/.intellisense/cli_config.yaml
default_settings:
  session_base_path: "C:/TESTRADE/intellisense_sessions"
  gui_enabled: true
  email_notifications: true
  backup_configs: true

collection_defaults:
  duration: "8h"
  background_mode: true
  quiet_mode: false
  symbols: ["AAPL", "MSFT", "GOOGL", "NVDA", "TSLA"]

analysis_defaults:
  engines: ["ocr", "price", "broker"]
  confidence_threshold: 80
  output_format: "gui"
  generate_report: true

optimization_defaults:
  safety_mode: "strict"
  test_before_apply: true
  backup_before_changes: true
  schedule_deployment: "market_close"

notification_settings:
  email_alerts: true
  desktop_notifications: true
  emergency_contact: "trader@company.com"
  alert_thresholds:
    high_impact_opportunity: 1000 # Dollar impact per day
    performance_regression: 5    # Percent performance drop
    experiment_failure: true
```

Configuration Management

Configuration Hierarchy

1. Global Configuration (GlobalConfig)

yaml

```
# C:/TESTRADE/config/global_config.yaml
```

```
application:
```

```
  name: "TESTRADE"
```

```
  version: "2.1.0"
```

```
  environment: "production"
```

```
intellisense:
```

```
  enabled: true
```

```
  mode: "background_collection" # background_collection, analysis, controlled_injection
```

```
  session_base_path: "C:/TESTRADE/intellisense_sessions"
```

```
# IntelliSense-specific configuration
```

```
collection_settings:
```

```
  max_session_duration: "24h"
```

```
  auto_start_collection: false
```

```
  correlation_log_compression: true
```

```
analysis_settings:
```

```
  default_engines: ["ocr", "price", "broker"]
```

```
  gui_auto_open: true
```

```
  confidence_threshold: 85
```

```
safety_settings:
```

```
  require_confirmation: true
```

```
  backup_configs: true
```

```
  max_position_exposure: 10000
```

```
  emergency_stop_enabled: true
```

```
# Your existing TESTRADE configuration continues normally
```

```
trading:
```

```
  symbols: ["AAPL", "MSFT", "GOOGL"]
```

```
  # ... rest of your normal config
```

2. Session Configuration

yaml

```
# Generated for each IntelliSense session
# C:/TESTRADE/intellisense_sessions/20241205_trading/session_config.yaml
session_info:
  name: "20241205_trading"
  type: "background_collection"
  created: "2024-12-05T09:25:00Z"
  duration: "8h"

data_collection:
  ocr_events:
    enabled: true
    capture_raw_frames: false
    include_confidence_scores: true

  price_events:
    enabled: true
    capture_level1: true
    capture_level2: false
    include_market_conditions: true

  broker_events:
    enabled: true
    capture_all_responses: true
    include_latency_metrics: true

correlation_settings:
  global_sequence_generation: true
  timestamp_precision: "nanoseconds"
  log_format: "jsonl"
  compression: true

output_settings:
  base_path: "./sessions/20241205_trading"
  file_naming:
    ocr_correlation: "ocr_correlation.jsonl"
    price_correlation: "price_correlation.jsonl"
    broker_correlation: "broker_correlation.jsonl"
```

3. Engine Configuration

yaml

```
# C:/TESTRADE/intellisense_sessions/analysis_config.yaml
intelligence_engines:
  ocr_intelligence:
    enabled: true
    real_services:
      snapshot_interpreter: true
      position_manager: true
    validation_settings:
      accuracy_threshold: 0.95
      confidence_threshold: 0.90
    performance_analysis:
      latency_analysis: true
      bottleneck_detection: true

  broker_intelligence:
    enabled: true
    real_services:
      order_repository: true
      position_manager: true
      lightspeed_broker: true
    validation_settings:
      order_state_validation: true
      position_state_validation: true
    analysis_focus:
      fill_processing: true
      ack_processing: true
      rejection_analysis: true

  price_intelligence:
    enabled: true
    real_services:
      price_repository: true
      market_data_service: true
    analysis_focus:
      feed_latency: true
      processing_efficiency: true
      data_quality: true
```

Configuration Management GUI

Configuration Editor

IntelliSense Configuration Manager	
<div> <div>GLOBAL SETTINGS</div> <div>SESSION TEMPLATES</div> </div>	
Mode: Background Collection	Daily Collection Template
Auto-start: <input type="checkbox"/> Enabled	Weekly Analysis Template
Base Path: C:/TESTRADE/...	Experiment Template
GUI Auto-open: <input checked="" type="checkbox"/> Yes	Custom Template...
<div>COLLECTION SETTINGS</div>	
OCR Events: <input checked="" type="checkbox"/> Enabled <input type="checkbox"/> Include Raw Frames	
Price Events: <input checked="" type="checkbox"/> Enabled <input checked="" type="checkbox"/> Level 1 <input type="checkbox"/> Level 2	
Broker Events: <input checked="" type="checkbox"/> Enabled <input checked="" type="checkbox"/> All Responses	
Correlation Logging:	
Precision: [Nanosecond ▼] Format: [JSONL ▼]	
Compression: <input checked="" type="checkbox"/> Enabled Encryption: <input type="checkbox"/> Enabled	
<div>SAFETY SETTINGS</div>	
Require Confirmation: <input checked="" type="checkbox"/> For all changes	
Backup Configs: <input checked="" type="checkbox"/> Before any modification	
Max Position: [\$10,000] emergency limit	
Emergency Contact: [trader@company.com]	
<div> [Test Configuration] [Save Changes] [Export] [Reset] </div>	

Configuration Templates

Daily Collection Template

yaml

```
# templates/daily_collection.yaml
template_name: "Daily Collection"
description: "Standard daily trading data collection"

session_config:
  type: "background_collection"
  duration: "8h"
  auto_start: "09:25"
  auto_stop: "16:05"

collection_settings:
  ocr_events: true
  price_events: true
  broker_events: true
  correlation_precision: "nanoseconds"

analysis_settings:
  auto_analyze_on_stop: true
  engines: ["ocr", "price", "broker"]
  generate_daily_report: true
  email_report_to: "trader@company.com"
```

Optimization Testing Template

yaml

```
# templates/optimization_testing.yaml
template_name: "Optimization Testing"
description: "Safe optimization testing session"

session_config:
  type: "analysis"
  safety_mode: "strict"

test_settings:
  simulation_mode: true
  confidence_threshold: 90
  test_duration: "30m"
  rollback_on_failure: true

validation_settings:
  require_manual_approval: true
  backup_before_changes: true
  emergency_stop_enabled: true
```

Controlled Injection Template

```
yaml
```

```
# templates/controlled_injection.yaml
template_name: "Controlled Injection Experiment"
description: "Safe controlled injection for optimization research"

session_config:
  type: "controlled_injection"
  isolation_mode: "full"

safety_settings:
  max_position_size: 10
  max_total_exposure: 1000
  auto_stop_timeout: "5m"
  paper_trading_mode: true

experiment_settings:
  bootstrap_required: true
  precision_timing: true
  detailed_logging: true
  gui_monitoring: true
```

Troubleshooting Guide

Common Issues and Solutions

1. Collection Not Starting

Symptom

```
bash

$ intellisense collect start --session "test"
ERROR: Failed to start collection - Enhanced components not available
```

Diagnosis

```
bash
```

```
# Check if IntelliSense is properly integrated
```

```
intellisense config validate
```

```
# Check if TESTRADE is running with IntelliSense support
```

```
intellisense status --detailed
```

Solutions

```
bash
```

```
# Solution 1: Ensure TESTRADE is running with IntelliSense mode
```

```
# Edit global_config.yaml:
```

```
intellisense:
```

```
  enabled: true
```

```
  mode: "background_collection"
```

```
# Solution 2: Restart TESTRADE with enhanced components
```

```
./testtrade.exe --intellisense-mode
```

```
# Solution 3: Check component integration
```

```
intellisense config test --verify-integration
```

2. Missing Data in Analysis

Symptom

```
bash
```

```
$ intellisense analyze start --session "./sessions/today"
```

```
WARNING: No broker events found in correlation logs
```

```
WARNING: Price events missing timestamps
```

Diagnosis

```
bash
```

```
# Check data collection quality
```

```
intellisense session inspect --path "./sessions/today"
```

```
# Verify correlation Log format
```

```
head -5 "./sessions/today/broker_correlation.jsonl"
```

Solutions

```
bash
```

```
# Solution 1: Verify all enhanced components are active
```

```
intellisense collect status --detailed
```

```
# Solution 2: Check data collection configuration
```

```
intellisense config show collection_settings
```

```
# Solution 3: Restart collection with verbose logging
```

```
intellisense collect start --session "debug_session" --verbose
```

3. GUI Not Opening

Symptom

```
bash
```

```
$ intellisense analyze start --gui
```

```
ERROR: GUI module not available or display not configured
```

Solutions

```
bash
```

```
# Solution 1: Install GUI dependencies
```

```
pip install intellisense[gui]
```

```
# Solution 2: Use X11 forwarding (if remote)
```

```
ssh -X user@trading-machine
```

```
intellisense analyze start --gui
```

```
# Solution 3: Use web interface instead
```

```
intellisense analyze start --web-interface --port 8080
```

```
# Open browser to http://localhost:8080
```

```
# Solution 4: Use CLI with HTML report
```

```
intellisense analyze start --output-format html
```

```
# Opens analysis_results.html in default browser
```

4. Performance Impact on Trading

Symptom

```
bash
```

TESTRADE trading latency increased by 2-3ms during collection

Diagnosis

```
bash
```

```
# Check collection overhead
```

```
intellisense collect status --performance-impact
```

```
# Monitor system resources
```

```
intellisense monitor resources --during-collection
```

Solutions

```
bash
```

```
# Solution 1: Reduce collection frequency
```

```
intellisense config set collection.sampling_rate 0.5 # Collect 50% of events
```

```
# Solution 2: Use asynchronous logging
```

```
intellisense config set collection.async_logging true
```

```
# Solution 3: Move to dedicated collection machine
```

```
intellisense collect start --remote-logging "analysis-machine:9999"
```

```
# Solution 4: Collect only during low-activity periods
```

```
intellisense collect start --schedule "09:30-10:00,15:30-16:00"
```

5. Controlled Injection Safety Issues

Symptom

```
bash
```

```
ERROR: Feedback isolation failed - OCR still reading real positions
```

```
WARNING: Test trades affecting live position calculations
```

Emergency Response

```
bash
```

```
# IMMEDIATE: Emergency stop all experiments  
intellisense experiment emergency-stop --all
```

```
# IMMEDIATE: Restore normal operation  
intellisense isolation restore --force
```

```
# IMMEDIATE: Verify system state  
intellisense status --safety-check
```

Prevention

```
bash
```

```
# Always test isolation before experiments  
intellisense isolation test --dry-run
```

```
# Use paper trading mode for initial tests  
intellisense experiment create --paper-trading-only
```

```
# Set conservative safety limits  
intellisense config set safety.max_position_size 1  
intellisense config set safety.auto_stop_timeout "60s"
```




Diagnostic Commands

System Health Check

```
bash
```

```
# Comprehensive system diagnostic  
intellisense diagnose --full-system
```

```
# Output example:
```

```
#  TESTRADE Integration: OK  
#  Enhanced Components: Active (OCR, Price, Broker)  
#  Data Collection: Running (847 events/min)  
#  Storage: 2.3GB free space  
#  Memory Usage: 73% (consider cleanup)  
#  Network Latency: 12ms (target: <5ms)
```

Performance Analysis

```
bash
```

```
# Analyze performance impact
```

```
intellisense performance analyze --baseline "/sessions/pre_intellisense" --current "/sessions
```

```
# Output:
```

```
# Performance Impact Analysis:
```

```
# OCR Latency: +0.2ms (+1.3%)
```

```
# Price Processing: +0.1ms (+0.8%)
```

```
# Broker Response: +0.3ms (+2.1%)
```

```
# Overall Impact: Minimal (within acceptable range)
```



Data Quality Check

```
bash
```

```
# Validate correlation data quality
```

```
intellisense data validate --session "/sessions/today"
```

```
# Output:
```

```
# Data Quality Report:
```

```
# ✓ OCR Events: 1,247 events, 99.8% correlated
```

```
# ✓ Price Events: 8,392 events, 100% correlated
```

```
# ⚠ Broker Events: 83 events, 94.2% correlated (5 missing timestamps)
```

```
#
```

```
# Recommendations:
```

```
# - Check broker interface logging configuration
```

```
# - Verify network stability during collection
```

Log File Locations

IntelliSense System Logs


```
bash
```

```
# Main system logs
```

```
C:/TESTRADE/intellisense_logs/
```

```
|─ system.log           # Main IntelliSense system log
|─ collection.log       # Data collection activities
|─ analysis.log         # Analysis engine logs
|─ experiment.log       # Controlled injection logs
└─ error.log            # Error and warning logs
```

```
# Session-specific logs
```

```
C:/TESTRADE/intellisense_sessions/{session_name}/
```

```
|─ session.log          # Session-specific activities
|─ performance.log      # Performance metrics during session
|─ safety.log           # Safety monitoring (for experiments)
└─ debug.log            # Debug information (if enabled)
```

Log Analysis Commands

```
bash
```

```
# Search for specific issues
```

```
intellisense logs search "ERROR" --last-24h
```

```
intellisense logs search "latency" --session "today" --verbose
```

```
# Generate Log summary
```

```
intellisense logs summary --date "2024-12-05"
```

```
# Export Logs for support
```

```
intellisense logs export --issue-id "INTL-001" --include-config
```

Best Practices

Daily Operations Best Practices

1. Pre-Market Setup (5 minutes)

bash

Morning checklist

- Check system status

intellisense status --health-check

- Verify available disk space

intellisense storage check --warn-below 10GB

- Start background collection

intellisense collect start --session "\$(date +%Y%m%d)" --background

- Verify collection is running

intellisense collect status --verify-all-components

2. During Market Hours

bash

Optional monitoring (minimal impact)

- Check status once per hour

intellisense status --quick

- Monitor **for** any alerts

intellisense alerts check --auto-clear-handled

- Keep collection running normally

No action needed - runs automatically

3. Post-Market Analysis (30 minutes)

bash

Evening optimization routine

□ Stop collection

```
intellisense collect stop --session "$(date +%Y%m%d)"
```

□ Run immediate analysis

```
intellisense analyze start --session "./sessions/$(date +%Y%m%d)" --gui
```

□ Review recommendations in GUI

Look for high-confidence, high-impact optimizations

□ Apply safe optimizations for tomorrow

```
intellisense optimize apply --safe-only --schedule "next-trading-day"
```

□ Test risky optimizations in simulation

```
intellisense optimize test --medium-risk --simulation-mode
```

4. Weekly Review (1 hour)

bash

Sunday evening weekly optimization

□ Analyze entire week's data

```
intellisense analyze weekly --sessions "./sessions/2024W*" --comparison-mode
```

□ Generate comprehensive report

```
intellisense report generate --template "weekly_optimization" --email-to "manager@company.com"
```

□ Plan upcoming week optimizations

```
intellisense plan generate --based-on "weekly_analysis" --conservative
```

□ Backup important sessions

```
intellisense backup create --sessions "./sessions/2024W*" --compress
```



Data Collection Best Practices

1. Collection Duration Guidelines

yaml

Optimal Collection Durations:

- Daily Analysis: 4-8 hours (full trading day)
- Weekly Analysis: 5 days (Monday-Friday)
- Parameter Testing: 1-2 hours (focused collection)
- Strategy Development: 2-3 days (sufficient sample size)

Minimum Viable Durations:

- OCR Optimization: 30 minutes (minimum 100 OCR events)
- Broker Analysis: 1 hour (minimum 20 order events)
- Price Feed Analysis: 15 minutes (minimum 1000 price events)

2. Data Quality Guidelines

yaml

Quality Thresholds:

- Event Correlation: >95% (events with proper timestamps)
- Missing Data: <5% (events without correlation data)
- Timestamp Precision: Nanosecond (for optimal analysis)
- Storage Efficiency: >80% compression ratio

Warning Indicators:

- Correlation Rate: <90% (investigate immediately)
- Large Timestamp Gaps: >10 seconds (check system performance)
- High Storage Usage: >1GB/hour (check compression settings)
- Memory Usage: >90% (consider reducing collection frequency)

3. Storage Management

bash

Automated cleanup policies

```
intellisense storage set-policy \  
  --keep-recent "30 days" \  
  --archive-older "90 days" \  
  --delete-older "1 year" \  
  --compress-after "7 days"
```

Monitor storage usage

```
intellisense storage monitor --alert-threshold 85% --email-alert
```

Manual cleanup when needed

```
intellisense storage cleanup \  
  --dry-run \  
  --keep-important-sessions \  
  --compress-old-sessions
```

Analysis Best Practices

1. Confidence Level Guidelines

yaml

Confidence Thresholds for Actions:

Auto-Apply: >95% confidence, Low risk

Manual Review: 80-95% confidence, Any risk level

Testing Required: 60-80% confidence, Any risk level

Research Only: <60% confidence, Document for future

Risk Level Definitions:

Low Risk: Parameter adjustments within 10% of current values

Medium Risk: Significant parameter changes or new configurations

High Risk: Architectural changes or experimental features

2. Optimization Priority Matrix

yaml

Priority 1 (Immediate): High Impact + High Confidence + Low Risk

- **Example:** OCR thread count optimization (15ms → 12ms, 94% confidence)
- **Action:** Apply immediately after validation

Priority 2 (Next Day): High Impact + Medium Confidence + Low Risk

- **Example:** Signal timeout adjustment (50ms → 35ms, 76% confidence)
- **Action:** Test in simulation, then apply

Priority 3 (Test Phase): Medium Impact + High Confidence + Medium Risk

- **Example:** Advanced OCR configuration changes
- **Action:** Design controlled experiment

Priority 4 (Research): Any Impact + Low Confidence + Any Risk

- **Example:** Experimental trading algorithms
- **Action:** Document for future research

3. Validation Requirements

bash

Always validate before applying optimizations

□ Run simulation **test**

intellisense optimize **test** --simulation-mode --duration "1h"

□ Check confidence interval

intellisense analyze confidence --minimum-threshold **80%**

□ Verify no negative side effects

intellisense analyze side-effects --check-all-metrics

□ Backup current configuration

intellisense config backup --tag "pre_optimization_\$(date +%Y%m%d)"

□ Apply with monitoring

intellisense optimize apply --monitor-for "24h" --rollback-on-regression

Safety Best Practices

1. Controlled Injection Safety Protocols

yaml

Pre-Experiment Checklist:

- ☐ Isolation system tested and verified
- ☐ Emergency stop procedures documented
- ☐ Maximum position limits configured
- ☐ Paper trading mode enabled (for initial tests)
- ☐ Real-time monitoring dashboard active
- ☐ Emergency contact notifications configured

During Experiment:

- ☐ Monitor experiment dashboard continuously
- ☐ Check safety metrics every 5 minutes
- ☐ Verify isolation is maintained
- ☐ Watch for unexpected position changes
- ☐ Confirm emergency stop is responsive

Post-Experiment:

- ☐ Verify complete system restoration
- ☐ Check all positions are as expected
- ☐ Review experiment logs for issues
- ☐ Document lessons learned
- ☐ Archive experiment data for future reference

2. Emergency Procedures

bash

Emergency Stop Protocol (memorize these commands)

EMERGENCY_STOP="intellisense experiment emergency-stop --all"

FORCE_RESTORE="intellisense isolation restore --force"

SAFETY_CHECK="intellisense status --safety-check --detailed"

Emergency contact procedure

- ☐ Execute emergency stop
- ☐ Verify system restoration
- ☐ Contact: trader@company.com (immediate)
- ☐ Contact: manager@company.com (within 15 minutes)
- ☐ Document incident for review

3. Risk Mitigation Strategies

yaml

Position Risk Mitigation:

- Never exceed 1% of account value in experiments
- Use paper trading for all initial tests
- Set automatic position limits
- Configure emergency liquidation triggers

System Risk Mitigation:

- Always backup configurations before changes
- Test optimizations in simulation first
- Implement gradual rollout of changes
- Monitor for performance regressions

Data Risk Mitigation:

- Encrypt sensitive correlation data
- Regular backup of important sessions
- Access control for experiment configuration
- Audit trails for all system changes

Advanced Usage Scenarios

Scenario 1: Multi-Strategy Optimization

Use Case

You run multiple trading strategies (scalping, momentum, mean reversion) and want to optimize each independently.

Setup

bash

```
# Create strategy-specific collection sessions
intellisense collect start --session "scalping_$(date +%Y%m%d)" --strategy-filter "scalping"
intellisense collect start --session "momentum_$(date +%Y%m%d)" --strategy-filter "momentum"
intellisense collect start --session "mean_reversion_$(date +%Y%m%d)" --strategy-filter "mean_r
```

Configuration

yaml

strategy_specific_config.yaml

collection_filters:

scalping:

symbols: ["AAPL", "MSFT", "GOOGL"] *# High-volume stocks*

time_windows: ["09:30-10:30", "15:30-16:00"] *# High volatility periods*

min_position_size: 100

momentum:

symbols: ["TSLA", "NVDA", "AMD"] *# Momentum stocks*

time_windows: ["10:00-15:30"] *# Main trading hours*

min_price_change: 0.5

mean_reversion:

symbols: ["SPY", "QQQ", "IWM"] *# ETFs*

time_windows: ["09:30-16:00"] *# Full day*

volatility_threshold: 0.02

analysis_configs:

scalping:

focus: "ultra_low_latency" *# Optimize for speed*

target_latency: "< 5ms"

momentum:

focus: "signal_accuracy" *# Optimize for accuracy*

target_accuracy: "> 85%"

mean_reversion:

focus: "execution_quality" *# Optimize for fills*

target_slippage: "< 0.02%"

Analysis Workflow

```
bash
```

```
# Analyze each strategy separately
```

```
intellisense analyze start --session "./sessions/scalping_*" --strategy "scalping" --gui  
intellisense analyze start --session "./sessions/momentum_*" --strategy "momentum" --gui  
intellisense analyze start --session "./sessions/mean_reversion_*" --strategy "mean_reversion"
```

```
# Compare strategies
```

```
intellisense compare strategies \  
  --sessions "./sessions/scalping_*,./sessions/momentum_*,./sessions/mean_reversion_*" \  
  --metrics "latency,accuracy,profitability" \  
  --output-format "comparative_report"
```

Scenario 2: Market Condition Adaptation

Use Case

Optimize trading parameters automatically based on market volatility, volume, and time of day.

Setup

```
bash
```

```
# Create market condition aware collection
```

```
intellisense collect start \  
  --session "adaptive_$(date +%Y%m%d)" \  
  --market-condition-tracking \  
  --volatility-monitoring \  
  --volume-monitoring
```

Market Condition Configuration

yaml

```
# market_adaptive_config.yaml
```

```
market_conditions:
```

```
  high_volatility:
```

```
    trigger: "VIX > 25 OR daily_range > 3%"
```

```
    optimization_focus: "risk_management"
```

```
    target_metrics:
```

- reduce_position_sizes
- increase_stop_losses
- faster_exit_signals

```
  low_volatility:
```

```
    trigger: "VIX < 15 AND daily_range < 1%"
```

```
    optimization_focus: "profit_maximization"
```

```
    target_metrics:
```

- increase_position_sizes
- wider_stop_losses
- longer_hold_times

```
  high_volume:
```

```
    trigger: "volume > 150% of 20_day_average"
```

```
    optimization_focus: "execution_speed"
```

```
    target_metrics:
```

- faster_order_placement
- reduced_latency
- immediate_fills

```
  low_volume:
```

```
    trigger: "volume < 50% of 20_day_average"
```

```
    optimization_focus: "execution_quality"
```

```
    target_metrics:
```

- larger_order_sizes
- patience_algorithms
- reduced_market_impact

```
time_based_optimization:
```

```
  market_open: # 9:30-10:00
```

```
    optimization_focus: "volatility_handling"
```

```
    target_latency: "< 3ms"
```

```
  midday: # 11:00-14:00
```

```
    optimization_focus: "efficiency"
```

```
    target_latency: "< 10ms"
```

```
market_close: # 15:30-16:00
  optimization_focus: "execution_certainty"
  target_latency: "< 5ms"
```

Adaptive Analysis

```
bash
```

```
# Analyze performance by market condition
intellisense analyze adaptive \
  --session "./sessions/adaptive_*" \
  --group-by "market_condition" \
  --optimize-per-condition \
  --generate-adaptive-rules

# Output: adaptive_optimization_rules.yaml
# - Conditional parameter sets for different market conditions
# - Automatic switching rules
# - Performance impact estimates
```

Scenario 3: Cross-Asset Class Optimization

Use Case

Optimize trading across multiple asset classes (equities, options, futures) with different latency and accuracy requirements.

Setup

yaml

cross_asset_config.yaml

asset_classes:

equities:

symbols: ["AAPL", "MSFT", "GOOGL"]

optimization_priority: "latency"

target_latency: "< 5ms"

collection_focus: ["ocr_speed", "broker_response"]

options:

symbols: ["AAPL_OPTIONS", "SPY_OPTIONS"]

optimization_priority: "accuracy"

target_accuracy: "> 95%"

collection_focus: ["signal_accuracy", "volatility_handling"]

futures:

symbols: ["ES", "NQ", "RTY"]

optimization_priority: "execution_quality"

target_slippage: "< 0.1 ticks"

collection_focus: ["order