IntelliSense User Guide

Complete Usage Manual for Your Trading Intelligence Platform

Table of Contents

- 1. Getting Started
- 2. <u>User Interface Options</u>
- 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

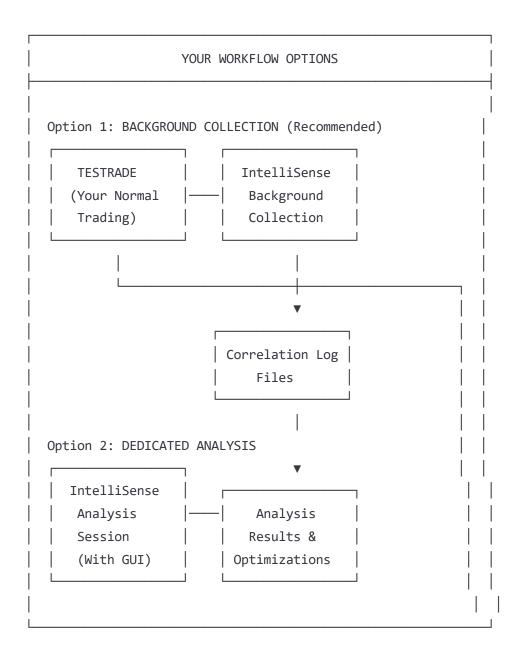
- Separate instance for replay and analysis
- Full GUI interface for interactive exploration

- 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

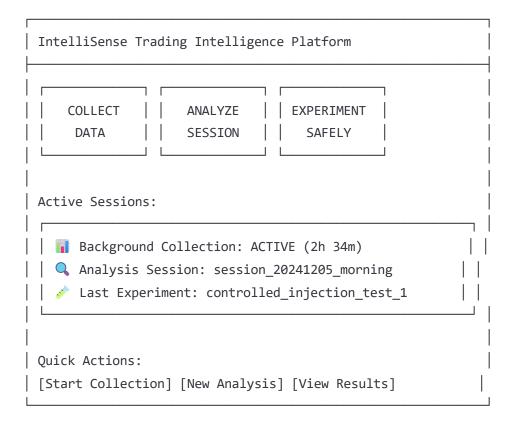
```
python
# GUI Application Entry Point
python -m intellisense.gui.main_dashboard
```

Features:

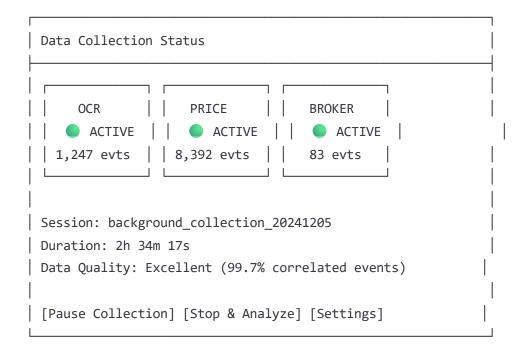
- **Session Management**: Create, load, and manage analysis sessions
- **Real-Time Monitoring**: Live performance metrics during data collection
- Interactive Analysis: Visual exploration of optimization opportunities
- Experiment Designer: GUI for setting up controlled injection experiments
- Results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

GUI Components

Main Dashboard



Data Collection Panel



Analysis Results Panel

```
Analysis Results - Session: morning_optimization
Performance Improvements Found:
  — OCR OPTIMIZATION —
 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
 [Apply Optimization] [Test Safely] [More Details]
BROKER OPTIMIZATION —
 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
  [Apply Optimization] [Test Safely] [More Details]
```

2. Command Line Interface (For Automation)

CLI Commands

```
# 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

```
# 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

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

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

```
# 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

```
# 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

```
# 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

```
# 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

```
# 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:

```
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

```
# Terminal 1: Your normal TESTRADE
./testrade.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

```
# 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

```
# 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</pre>
```

Analysis Session Resources

```
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)

```
# 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

2. Real-Time Monitoring

```
Live Performance Monitor
OCR PERFORMANCE — BROKER PERFORMANCE —
Current Latency: 15.3ms
                      Response Time: 8.7ms
 Target: < 12ms
                      | | Target: < 10ms
                        Status: 🔽 GOOD
 OPPORTUNITY
                      Recent Orders:
 Last 100 frames:
                      AAPL BUY 100 (7.2ms)

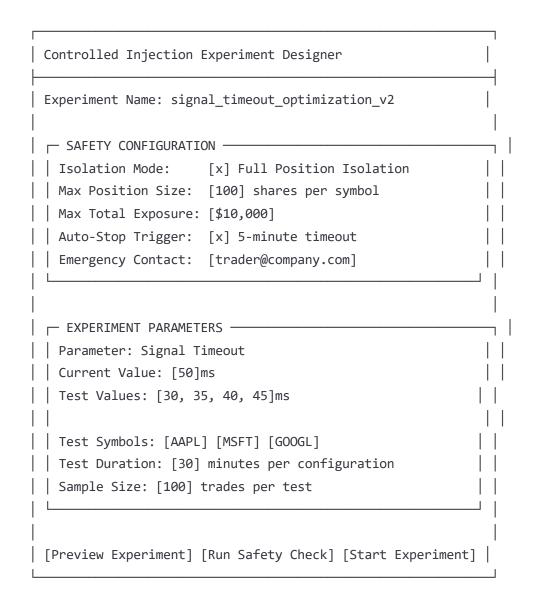
    PRICE FEED PERFORMANCE

Feed Latency: 2.1ms
                    Events/sec: 847
Status: EXCELLENT Last Update: 09:34:22.147
```

3. Analysis Results Viewer

```
Analysis Results - Session: 20241205_morning
© OPTIMIZATION OPPORTUNITIES FOUND: 3
HIGH IMPACT —
♦ OCR Thread Optimization
    Current: 15.3ms avg → Optimized: 12.1ms (-3.2ms)
    Confidence: 94% | Daily Impact: $1,247
    [Apply Now] [Test First] [More Details]
__ MEDIUM IMPACT _____
Signal Timeout Adjustment
    Current: 50ms → Optimized: 35ms (-15ms)
    Confidence: 76% | Daily Impact: $423
    [Apply Now] [Test First] [More Details]
EXPERIMENTAL ----
 Advanced OCR Configuration
   Potential improvement: 2.1ms
   Confidence: 52% | Requires Testing
   [Design Experiment] [More Details]
```

4. Experiment Designer



GUI Navigation

Main Menu Structure

File
├─ New Session
│ ├── Background Collection
│
│ └─ 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

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:

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

```
# Start background collection
intellisense collect start [options]
  --session-name TEXT
                             Session name
                            Duration (e.g., "8h", "all_day")
  --duration TEXT
  --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]
                            Specific session status
  --session-name TEXT
  --all
                            All active sessions
  --detailed
                            Detailed statistics
```

Analysis Commands

```
hash
# 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