Numba WFO

Walk-Forward Optimization for Systematic Trading Strategy Discovery

58M+ parameter combinations. 4 overlapping market regimes. Only the robust survive.

58M+	6	4	~15 min
Combinations Tested	Strategy Variants	WFO Phases	Runtime (32 threads)

A research-grade pipeline that stress-tests millions of parameter combinations across overlapping market regimes, surfaces the robust ones, and exports refinement-ready configurations for TradeStation.

The output is not a single "best" backtest. It is a map of robust parameter regions — plateaus where many nearby combinations all perform well — ready for live deployment.

The Problem

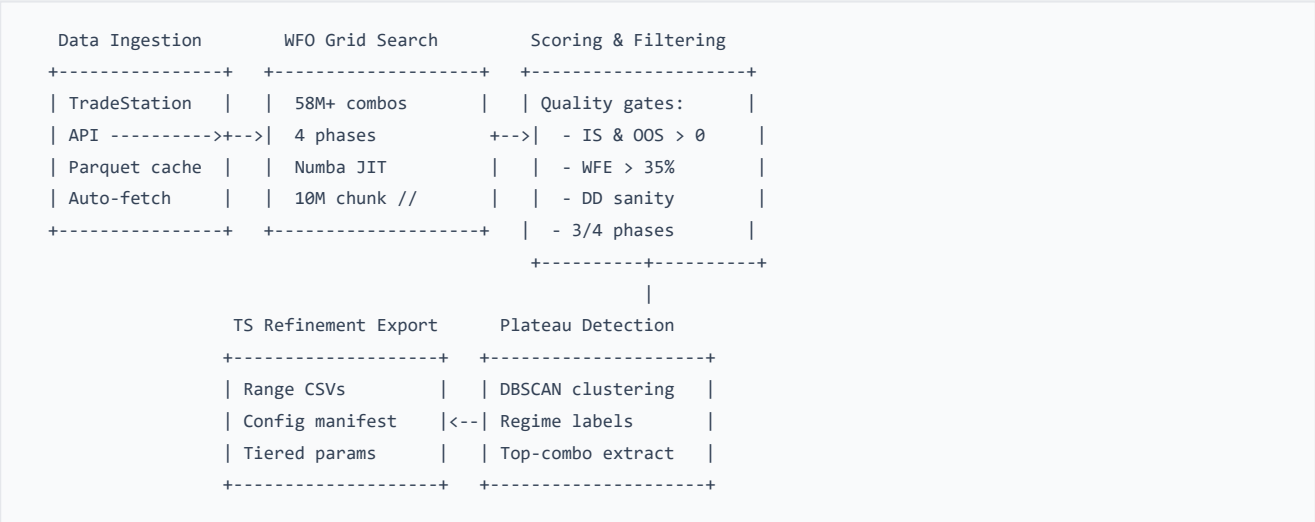
Manual strategy optimization is slow, brittle, and biased. Most backtests find parameters that worked in-sample but collapse out-of-sample. Traders waste weeks tweaking knobs on a single symbol, with no systematic way to separate signal from noise.

Walk-forward optimization solves this — but brute-force WFO over millions of combinations is computationally prohibitive in Python, and TradeStation's built-in optimizer can't search at this scale.

The Solution

Numba WFO replaces guesswork with exhaustive, parallelized parameter search. Every combination is tested across 4 overlapping time windows. Only parameters that prove stable across multiple independent market regimes survive.

End-to-End Pipeline



Stage 1 Data Ingestion

TradeStation API with OAuth 2.0, incremental Parquet cache, auto-fetch with graceful fallback on holidays/weekends.

Stage 2 WFO Grid Search

58M+ parameter combinations tested across 4 overlapping IS/OOS phases. Numba JIT kernels split the work into 10M-element parallel chunks.

Stage 3 Scoring & Filtering

Three quality gates (profitability, WFE > 35%, drawdown sanity) per phase. Cross-phase stability requires passing 3 of 4 windows.

Stage 4 Plateau Detection

DBSCAN clustering on the (atr_multiplier, atr_buy_multiplier) landscape. Identifies regions of robust performance, not isolated peaks.

Stage 5 TS Refinement Export

Generates tiered parameter range CSVs and a config manifest for TradeStation's Phase 5 fine-grained optimizer.

Strategy Library

Six JIT-compiled strategy kernels built on the **ATR-D v1.5** architecture. Each is a self-contained Numba function that processes raw OHLC arrays and returns trade-level PnL.

Variant	Direction	Key Signals	Inputs	Params
atr_d_l	Long-only	ATR trailing stop + buy-touch	1 sym	11
atr_d_ls	Long/Short	+ short mult, trend veto	1 sym	16
atr_d_hma_l	Long-only	+ Hull MA crossover	1 sym	13
atr_d_hma_ls	Long/Short	+ HMA + shorts	1 sym	14
atr_d_duals_l	Long-only	Data1 price, Data2 BB Width	2 sym	11
atr_d_duals_ls	Long/Short	Dual-symbol + short logic	2 sym	16

> Automatic strategy expansion — requesting atr_d_duals_ls also runs duals_l, ls, and l variants, so you can see whether dual-symbol and short-side logic actually improve results.

Core Mechanics

Adaptive Trailing Stop

The stop adjusts dynamically to volatility:

```
stop = highest_since_entry - ATR(period) x multiplier x bb_tightening
```

- **Bollinger Band tightening**: When BB Width drops below a volatility threshold, the stop narrows — protecting profits during low-vol squeezes
- **Three scaling modes**: Linear, square-root, and quadratic tightening curves
- **Max stop distance gate**: A 70-bar ATR baseline caps how wide the stop can drift, preventing catastrophic single-trade losses

Buy-Touch Re-Entry

After a stop-out, the strategy re-enters when price pulls back to a configurable ATR-based threshold — capturing mean-reversion bounces without chasing momentum.

Short-Side Logic (LS Variants)

- **Short multiplier** scales the ATR period and multiplier for shorts (e.g. 0.6x = tighter short stops)
- **Trend veto** blocks short entries when the EMA slope is below a threshold
- **Entry modes**: Flat-only transitions, long-priority, or full bidirectional reversals

Hull Moving Average (HMA Variants)

Fast/slow HMA crossover gates entries. Re-entry via buy-touch requires the HMA trend to still be favorable — no re-entering against the trend.

Dual-Symbol (DualS Variants)

Trades a leveraged ETF (e.g. TQQQ) using its own price for ATR stops, but computes BB Width from the underlying index (e.g. QQQ) — producing more stable volatility signals on inherently noisy leveraged instruments.

Walk-Forward Optimization

4-Phase Overlapping Windows

Each phase is an independent continuous backtest with an IS/OOS split. Overlapping windows ensure every year of data participates in multiple roles (in-sample and out-of-sample):

```
Phase 1:  |--- 6y IS ---|--- 4y OOS ---|                 10y -> 5y ago
Phase 2:       |--- 5y IS ---|--- 5y OOS ---|            8y -> 3y ago
Phase 3:            |--- 5y IS ---|--- 5y OOS ---|       6y -> 1y ago
Phase 4:                 |--- 4y IS ---|--- 4y OOS ---|   4y -> present
          ------------------------------------------------>
          past                                    today
```

- **Continuous capital flow**: IS profits compound into OOS, matching real-world equity curves
- **100-bar warmup**: Every phase initializes indicators before the first trade
- **Open position handoff**: Trades spanning the IS/OOS boundary are marked-to-market and carried forward

Chunk-Parallel Execution

The 58M+ combination space is split into 10M-element chunks, each dispatched to a Numba @njit(parallel=True) kernel. Typical runtime: 8-12 minutes on a 32-thread machine.

Scoring System (v4)

Quality Gates (Per Phase)

Every combination must clear three filters in each phase it participates in:

Gate	Rule	Purpose
Profitability	IS PnL > 0 AND OOS PnL > 0	Eliminates curve-fit losers
Walk-Forward Efficiency	OOS$ / IS$ > 0.35	OOS retains 35%+ of IS edge
Drawdown Sanity	OOS DD < OOS PnL, OOS DD < 3x IS DD	Catches regime-blown combos

Cross-Phase Stability

A combination must pass quality gates in 3 of 4 phases to be considered stable. This is the primary robustness filter — one bad phase is tolerated, but consistent failure is not.

If no combinations qualify at 3/4, the threshold auto-relaxes to 2/4 with a warning.

Stability Score

```
Stability = avg_rank x phase_bonus x consistency_factor
```

Component	Formula	Effect
avg_rank	Mean percentile rank across valid phase	Higher OOS PnL = higher rank
phase_bonus	1 + (phases_valid - 1) x 0.1	4-phase: 1.3x; 3-phase: 1.2x
consistency_factor	1 / (1 + stdev(ranks) / 20)	Penalizes erratic phase swings

The top 2M combinations by stability score are returned as candidates.

Plateau Detection

> Rather than picking a single "best" parameter set, the system identifies regions of parameter space where many nearby combinations all perform well. Plateaus are inherently more robust than isolated peaks.

How It Works

1. Aggregate all candidates into (atr_multiplier, atr_buy_multiplier) cells
2. Filter cells with fewer than 50 stable combinations
3. Cluster the surviving cells using DBSCAN with automated hyperparameter search
4. Classify each cluster's stop regime (see table below)
5. Extract the top 10% of combinations from each plateau for refinement

Regime	ATR Multiplier	Character
TIGHT	< 5	Aggressive stops, more trades, tighter risk
MEDIUM	5 – 12	Balanced risk/reward
WIDE	> 12	Loose stops, fewer trades, trend-following

TradeStation Integration

Range Export

For each plateau, the system generates a phase5_refine_*.csv with parameter ranges ready for TradeStation's optimizer:

Tier	Parameters	Treatment
Tier 1 (search)	ATR periods, multipliers, BB deviation, HMA le	Min/max/step from plateau bounds
Tier 2–3 (fixed)	Scaling method, entry mode, vol threshold	Locked at median

Refinement Analysis

After TradeStation runs Phase 5 refinement, run_analysis.py ranks the final results across six metrics with per-strategy rank-sum composite scoring:

- **MAR Ratio** — CAGR / Max Drawdown
- **Sharpe Ratio** — risk-adjusted return
- **Sortino Ratio** — downside-risk-adjusted return
- **Profit Factor** — gross profit / gross loss
- **Profit / Drawdown** — net profit / max drawdown
- **Robustness** — % of profitable combinations in the source plateau

Output: a formatted Excel workbook with conditional formatting, ready for final selection.

Data Management

Auto-Fetch

run_wfo.py automatically checks data coverage before every run:

- Reads `{SYMBOL}_1day.meta.json` to determine cached date range
- If data is missing or stale, calls `ParquetStore.get()` to incrementally download only the gap
- Gracefully falls back to cached data if the API is unavailable (weekends, holidays, network issues)
- `--no-fetch` disables auto-fetch for offline/air-gapped environments

Incremental Parquet Cache

- **Cache hit**: meta.json covers the requested range — instant return
- **Partial miss**: Only fetches the pre-gap or post-gap, merges with existing data
- **Full miss**: Downloads the entire range, writes parquet + meta.json

Key Design Decisions

Decision	Rationale
Continuous IS→OOS backtest	Matches TradeStation's single-simulation model; capital compounds naturally.
Ring-buffer indicators	O(1) per-bar updates, cache-friendly memory access for Numba JIT kernels.
2D DBSCAN on multipliers	Interpretable axes (stop width vs. entry sensitivity), directly actionable for refinement.
v4 quality-gate scoring	Earlier versions used top-20% cutoffs that missed high-quality regions; v4 uses hard filters + cross-phase stability.
Strategy expansion	Automatically answers "does adding shorts/dual-symbol/HMA actually help?" without separate manual runs.
Yesterday as fetch end-date	Today's daily bar isn't available until after market close; avoids API 404s on holidays and weekends.

Quick Start

```
# Single symbol, long-only — auto-downloads, runs WFO, plateaus, exports ranges
python run_wfo.py --symbol SMH --strategy atr_d_l --grid phase1_slim

# Dual-symbol with full expansion (runs 4 variants automatically)
python run_wfo.py --symbol TQQQ --symbol2 QQQ --strategy atr_d_duals_ls

# Re-score existing results with updated scoring logic
python run_wfo.py --symbol SMH --rescore --wfo-dir results/SMH/wfo/atr_d_l

# Analyze TradeStation refinement output
python run_analysis.py --symbol SMH --top 50

# Manual data download (minute bars, specific date range)
python download_data.py --symbols SMH QQQ --timeframe 1min --start 2016-01-01 --end 2026-02-14
```

Output Structure

```
results/SMH/
+-- wfo/
|   +-- atr_d_l/phase1/         # Binary cache: is_pnl.npy, oos_pnl.npy, ...
|   +-- atr_d_l/phase2/
|   +-- atr_d_l/phase3/
|   +-- atr_d_l/phase4/
|   +-- atr_d_ls/phase1/ ... phase4/
|   +-- ...
+-- candidates_atr_d_l_phase1_slim.csv       # Stable combos with scores
+-- candidates_atr_d_ls_phase1_slim.csv
+-- phase5_refine_ATR-D-v1.5-L_rank1.csv     # TS refinement ranges
+-- phase5_refine_ATR-D-v1.5-L_rank2.csv
+-- phase5_refine_ATR-D-v1.5-LS_rank1.csv
+-- phase5_refine_config.csv                  # Combined config
+-- Refinement_Analysis_SMH.xlsx              # Final ranked results
```

Requirements

- Python 3.10+
- Numba, NumPy, Pandas, SciPy, scikit-learn, PyArrow
- TradeStation API credentials (for data download)
- TradeStation desktop (for Phase 5 refinement)