Market Microstructure & Algorithmic Trading

ORIE 5259

Rethyam, Adarsh and Atharwa

Algorithm Overview

Buy-side trades are executed when there's strong evidence of upward pressure and low execution costs, while sell-side trades are triggered under signs of bearish sentiment and similarly favorable liquidity. By requiring alignment across multiple indicators, the approach ensures high-confidence execution.

The strategy utilizes six microstructure-based signals—volume imbalance, spread, momentum, trade intensity, volume curve deviation, and aggression ratio.

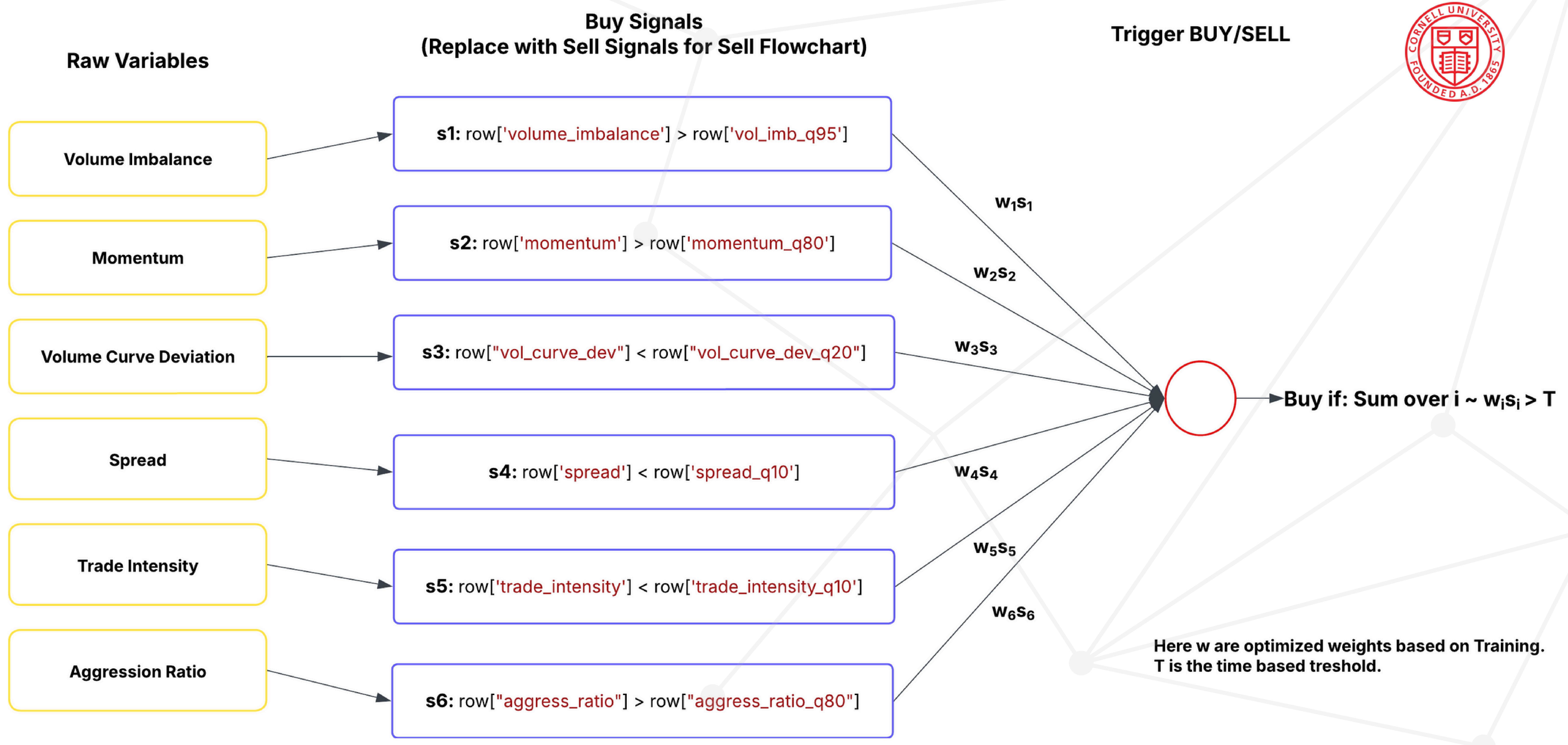
The strategy incorporates weighted signals, where each signal's contribution is adjusted based on its impact on execution quality for a given stock during training, allowing the strategy to prioritize the most predictive indicators. Additionally, it employs time-based thresholds to control when the strategy actually trades within a given minute, dividing the minute into distinct zones to identify the most favorable interval for execution.

Note: We use TWAP as the benchmark, which executes at the first available opportunity within each minute.

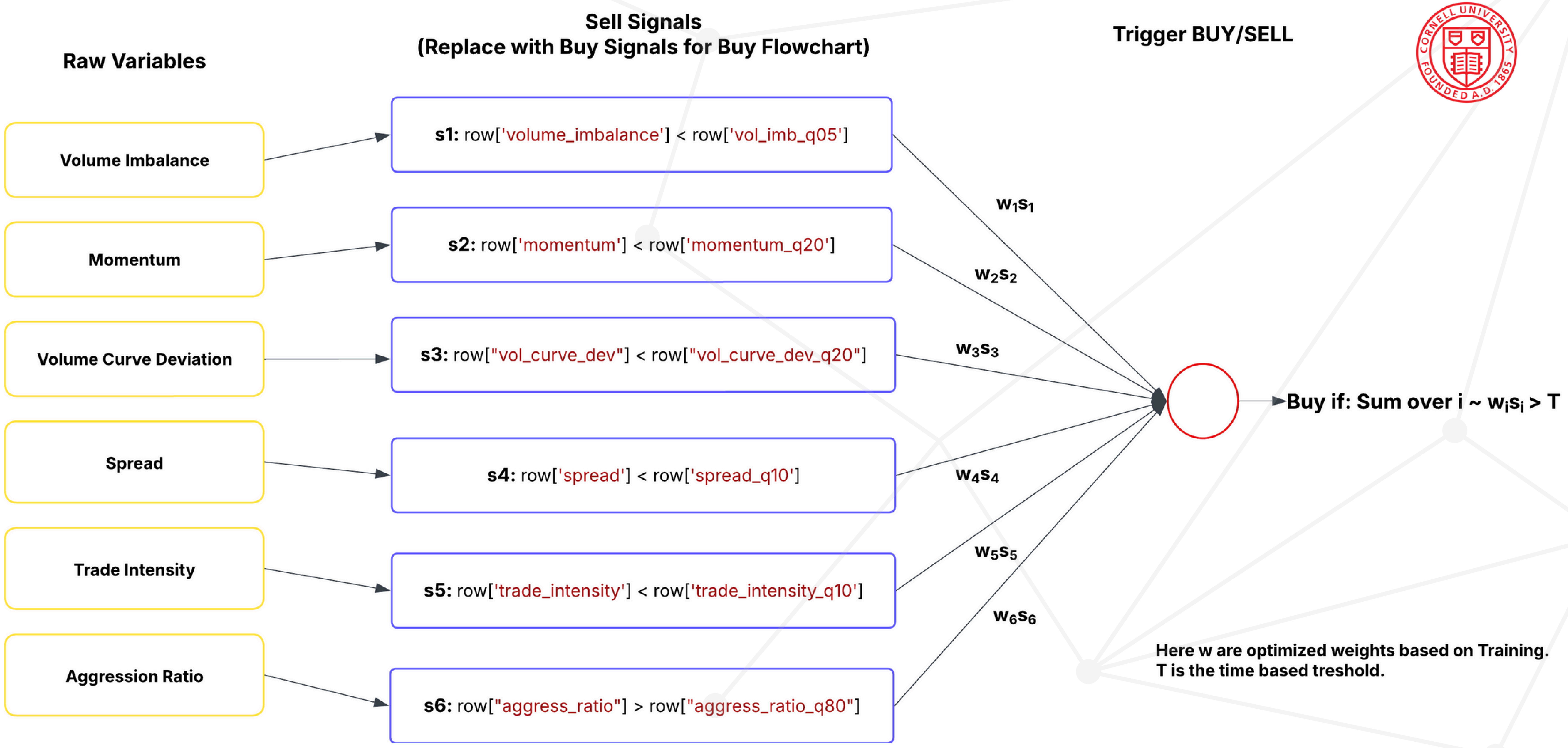
Variables and Signals

Volume Imbalance	captures real-time buying or selling pressure in the order book.
Momentum	detects short-term directional trends, helping time trades with favorable price movement.
Volume Curve Deviation	flags unusual volume spikes or drops relative to recent activity.
Spread	reflects execution cost, guiding trades toward tighter and more efficient pricing.
Trade Intensity	measures market activity levels to avoid trading in overly noisy periods.
Aggression Ratio	signals urgency from market participants, indicative of informed trading behavior.

Each variable, when converted to a signal, captures a key execution factor: Volume Imbalance and Aggression Ratio reflect pressure and urgency, Momentum captures trend, Volume Curve Deviation signals abnormal activity, and Spread and Trade Intensity account for cost and market conditions. ***Continued on next page.***



Raw Variables
Buy Signals
(Replace with Sell Signals for Sell Flowchart)
Trigger BUY/SELL
Volume Imbalance
Momentum
Volume Curve Deviation
Spread
Trade Intensity
Aggression Ratio
s1: row['volume_imbalance'] > row['vol_imb_q95']
s2: row['momentum'] > row['momentum_q80']
s3: row["vol_curve_dev"] < row["vol_curve_dev_q20"]
s4: row['spread'] < row['spread_q10']
s5: row['trade_intensity'] < row['trade_intensity_q10']
s6: row["aggress_ratio"] > row["aggress_ratio_q80"]
w_1s_1
w_2s_2
w_3s_3
w_4s_4
w_5s_5
w_6s_6
Buy if: Sum over i ~ w_is_i > T
Here w are optimized weights based on Training.
T is the time based treshold.


Raw Variables
Sell Signals
(Replace with Buy Signals for Buy Flowchart)
Trigger BUY/SELL
Volume Imbalance
Momentum
Volume Curve Deviation
Spread
Trade Intensity
Aggression Ratio
s1: row['volume_imbalance'] < row['vol_imb_q05']
s2: row['momentum'] < row['momentum_q20']
s3: row["vol_curve_dev"] < row["vol_curve_dev_q20"]
s4: row['spread'] < row['spread_q10']
s5: row['trade_intensity'] < row['trade_intensity_q10']
s6: row["aggress_ratio"] > row["aggress_ratio_q80"]
w_1s_1
w_2s_2
w_3s_3
w_4s_4
w_5s_5
w_6s_6
Buy if: Sum over i ~ w_is_i > T
Here w are optimized weights based on Training.
T is the time based treshold.

Dynamic Quantile Tresholds

Buy Signals
(Replace with Sell Signals for Sell Flowchart)

s1: row['volume_imbalance'] > row['vol_imb_q95']

s2: row['momentum'] > row['momentum_q80']

s3: row["vol_curve_dev"] < row["vol_curve_dev_q20"]

s4: row['spread'] < row['spread_q10']

s5: row['trade_intensity'] < row['trade_intensity_q10']

s6: row["aggress_ratio"] > row["aggress_ratio_q80"]

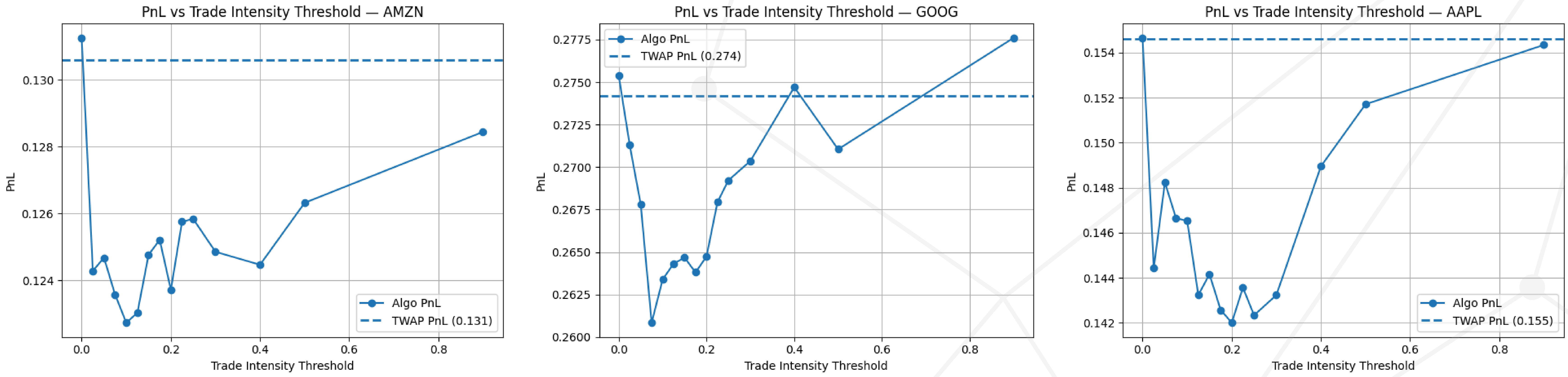
The strategy uses quantile-based thresholds computed over a 5-minute rolling window to determine whether a signal is strong enough to trigger a trade, as shown in the figure on the left!!

A quantile represents a statistical cutoff—for example, the 95th percentile indicates values higher than 95% of past observations.

By comparing real-time values to these thresholds, the strategy adapts to changing market conditions.

Note: While this strategy uses quantiles over a 5-minute lookback to balance stability and responsiveness, one could use online learning instead if the goal is to emphasize very recent signals.

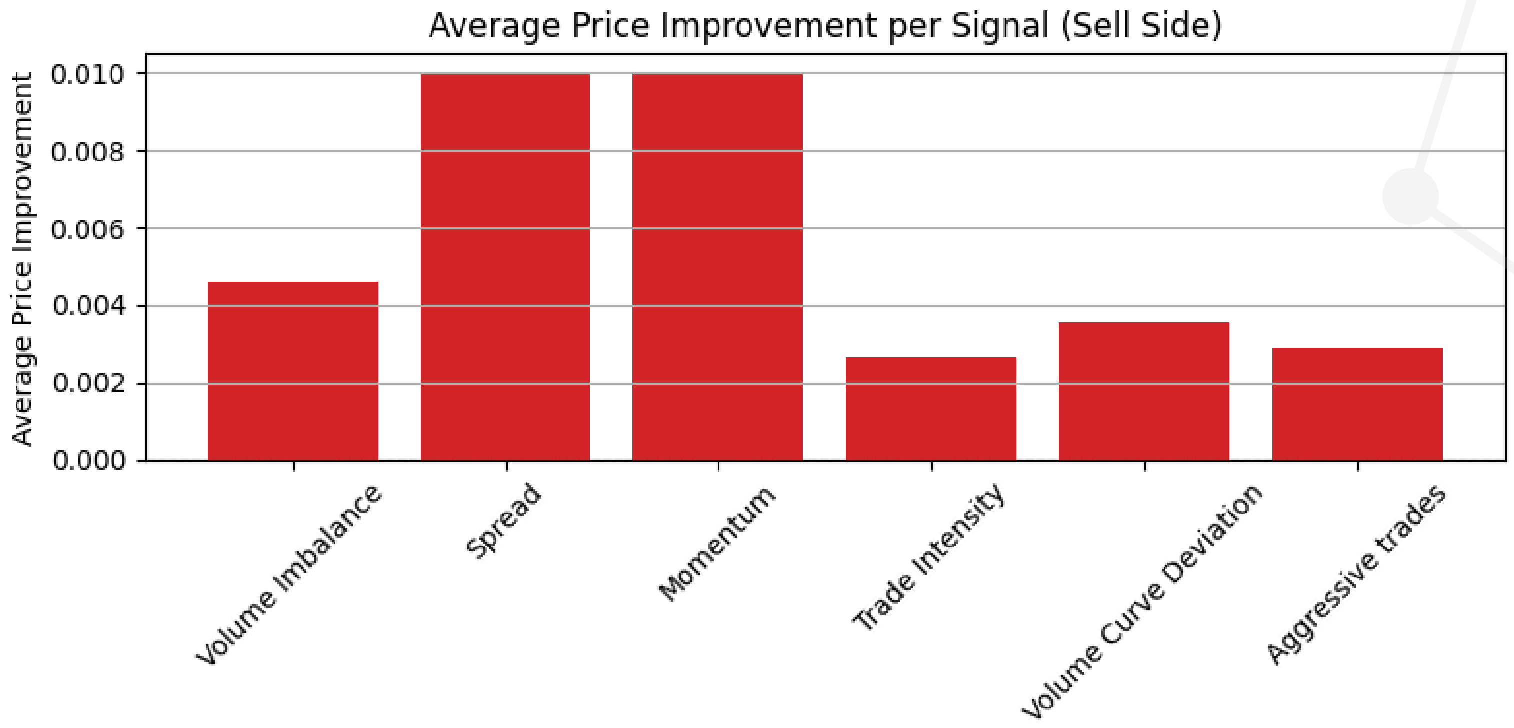
How to decide the Tresholds?



For example, in one of our signals, we use the condition **row['trade_intensity'] < row['trade_intensity_q10']**

But how did we choose the 10th percentile (q10) as the threshold? We analyzed plots showing how the P&L varied across different quantile cutoffs for each signal, and combined those insights with domain intuition to select the thresholds that consistently aligned with better execution outcomes.

Why Weighted Signals?

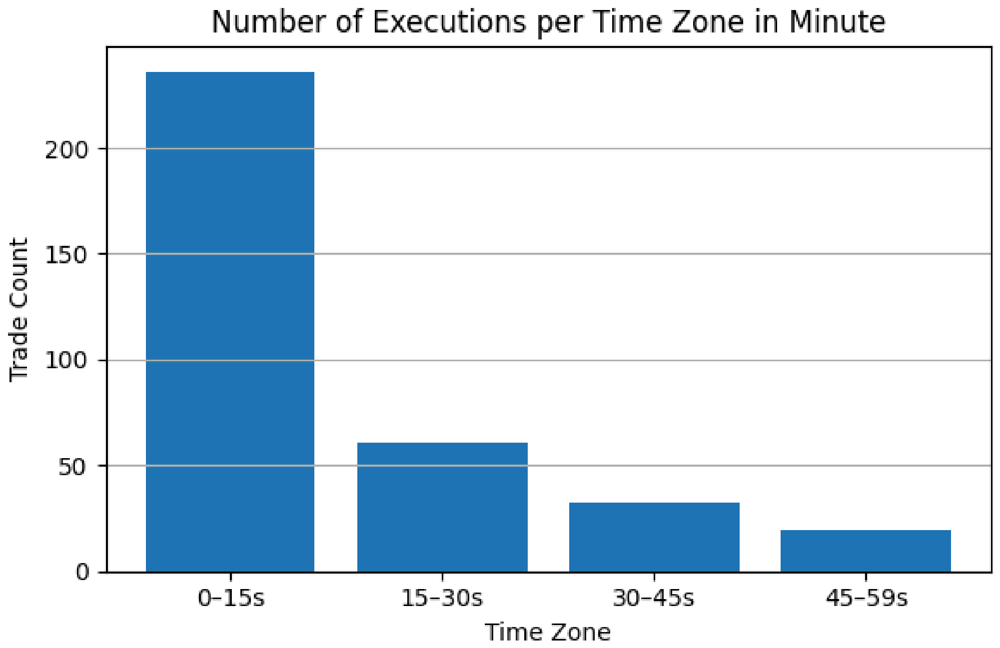


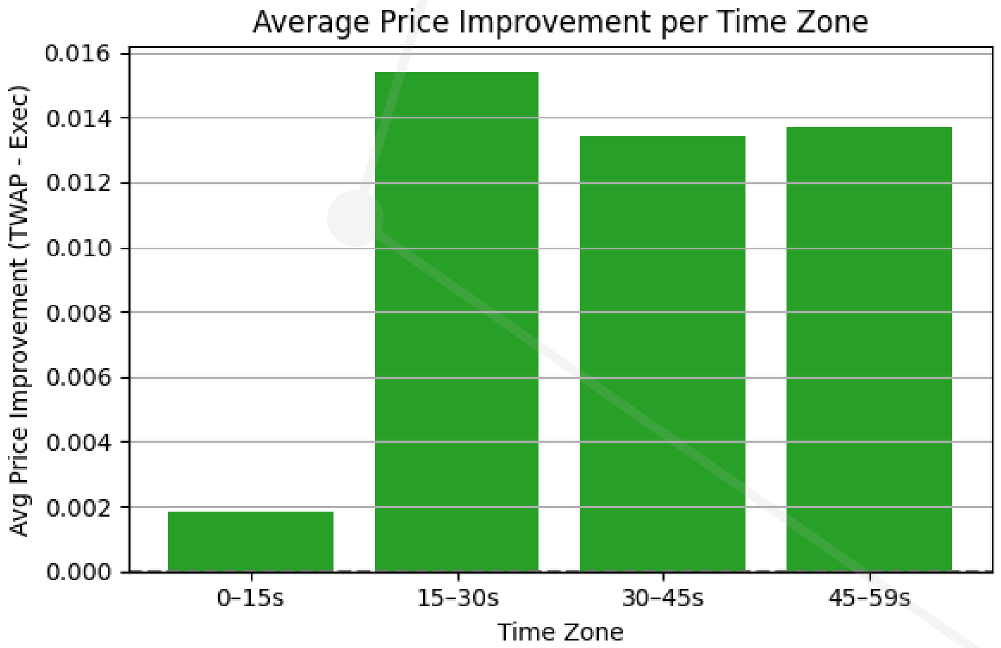
The plot highlights that signals contribute unevenly to price improvement when included equally weighted in trade decisions. This variation suggests that equal weighting can dilute the effect of stronger signals and amplify weaker ones. To address this, we adopt a weighted signal approach, where weights for each signal are optimized during the training process for each individual stock.

Weight Optimizing Process discussed ahead!

Plots: AAPL Sell Side (Left) and MSFT Buy Side (Right)

Time Based Heuristic Division



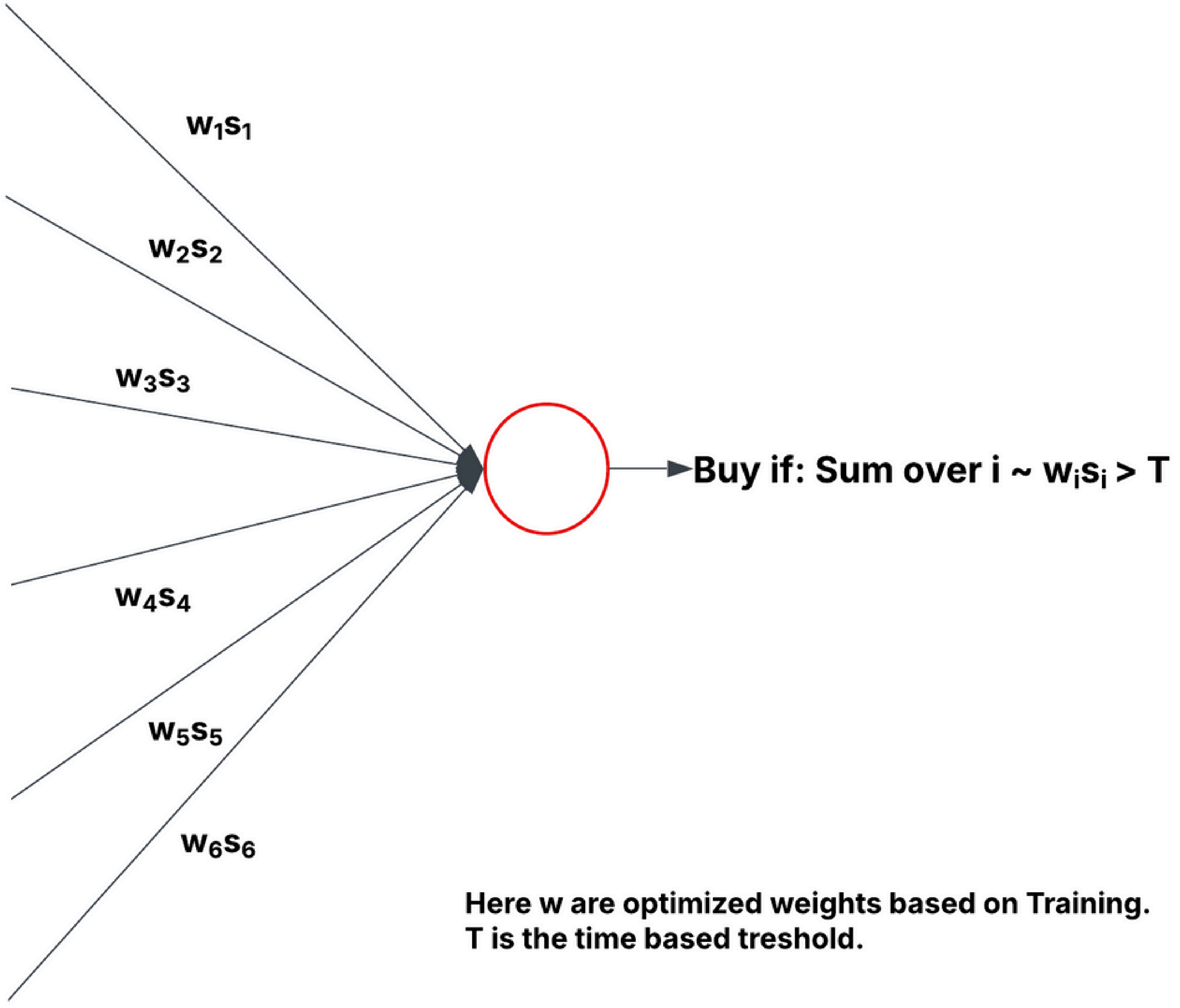


Before we jump into weight optimization, let's quickly go over the heuristic time-based breakdown as well!

Now we discuss the time-based division within a minute for execution. As seen in the plots above, the number of trades and their impact vary significantly across the different time zones within each minute.

To address this, we first optimize signal weights using equal thresholds (T) across all zones. Then, based on insights from these plots, we manually adjust the thresholds for each zone and re-optimize the weights accordingly. This manual adjustment process introduces a degree of subjectivity, which we acknowledge as a limitation and discuss further ahead.

Weight Optimization



Once the time-based thresholds are finalized, we re-optimize the signal weights to reflect the updated execution dynamics.

We use the *Optuna library* in Python—Optuna applies Bayesian optimization techniques, particularly Tree-structured Parzen Estimators, to explore the search space and focus on the most promising combinations. Unlike grid search, which tests every possible configuration exhaustively, Optuna efficiently narrows down the options.

Each signal can take a weight from the set {0, 1, 2, 3, 4}, resulting in nearly 15,000 combinations if we used grid search—but Optuna finds optimal solutions much faster.

Implementation Limitations

One limitation of our approach is that we only explored *whole-number weights for signals in the range {0, 1, 2, 3, 4} due to time constraints and the large number of possible combinations.* This coarse granularity likely leaves room for further optimization through finer weight tuning.

Secondly, the time-based thresholds were manually adjusted based on visual analysis of trade distributions across intraminute zones. *While intuitive and guided by observed patterns, this approach lacks scalability and consistency in dynamic environments* and could be improved through automated, data-driven optimization.

Results - IS (Left) and OS (Right)

Ticker	Execution	Improvement	Improvement Code
GOOG	Buy	0.084179	np.mean(twap_prices - exec_prices)
GOOG	Sell	0.05268	np.mean(exec_prices - twap_prices)
Mean Difference (Algo)		0.1373487	np.mean(buy_prices_algo - sell_prices_algo)
Mean Difference (TWAP)		0.27420749	np.mean(buy_prices_twap - sell_prices_twap)
Percent Reduction		49.91066801	
AAPL	Buy	0.050597	np.mean(twap_prices - exec_prices)
AAPL	Sell	0.051932	np.mean(exec_prices - twap_prices)
Mean Difference (Algo)		0.05207386	np.mean(buy_prices_algo - sell_prices_algo)
Mean Difference (TWAP)		0.15460227	np.mean(buy_prices_twap - sell_prices_twap)
Percent Reduction		66.31753208	
MSFT	Buy	0.006293	np.mean(twap_prices - exec_prices)
MSFT	Sell	0.004137	np.mean(exec_prices - twap_prices)
Mean Difference (Algo)		-0.00028736	np.mean(buy_prices_algo - sell_prices_algo)
Mean Difference (TWAP)		0.01014368	np.mean(buy_prices_twap - sell_prices_twap)
Percent Reduction		102.8328969	
AMZN	Buy	0.037595	np.mean(twap_prices - exec_prices)
AMZN	Sell	0.026275	np.mean(exec_prices - twap_prices)
Mean Difference (Algo)		0.06671554	np.mean(buy_prices_algo - sell_prices_algo)
Mean Difference (TWAP)		0.13058651	np.mean(buy_prices_twap - sell_prices_twap)
Percent Reduction		48.91084845	
INTC	Buy	0.0047	np.mean(twap_prices - exec_prices)
INTC	Sell	0.003862	np.mean(exec_prices - twap_prices)
Mean Difference (Algo)		0.00161677	np.mean(buy_prices_algo - sell_prices_algo)
Mean Difference (TWAP)		0.01017964	np.mean(buy_prices_twap - sell_prices_twap)
Percent Reduction		84.11761123	

Ticker	Execution	Improvement	Improvement Code
GOOG	Buy	0.063181818	np.mean(twap_prices - exec_prices)
GOOG	Sell	0.057954545	np.mean(exec_prices - twap_prices)
Mean Difference (Algo)		0.06840909	np.mean(buy_prices_algo - sell_prices_algo)
Mean Difference (TWAP)		0.18954545	np.mean(buy_prices_twap - sell_prices_twap)
Percent Reduction		63.90887252	
AAPL	Buy	0.043076923	np.mean(twap_prices - exec_prices)
AAPL	Sell	-0.007179487	np.mean(exec_prices - twap_prices)
Mean Difference (Algo)		0.05871795	np.mean(buy_prices_algo - sell_prices_algo)
Mean Difference (TWAP)		0.09461538	np.mean(buy_prices_twap - sell_prices_twap)
Percent Reduction		37.94037502	
MSFT	Buy	0.003488372	np.mean(twap_prices - exec_prices)
MSFT	Sell	0.004186047	np.mean(exec_prices - twap_prices)
Mean Difference (Algo)		0.00255814	np.mean(buy_prices_algo - sell_prices_algo)
Mean Difference (TWAP)		0.01023256	np.mean(buy_prices_twap - sell_prices_twap)
Percent Reduction		75	
AMZN	Buy	0.0098	np.mean(twap_prices - exec_prices)
AMZN	Sell	0.0288	np.mean(exec_prices - twap_prices)
Mean Difference (Algo)		0.0558	np.mean(buy_prices_algo - sell_prices_algo)
Mean Difference (TWAP)		0.0944	np.mean(buy_prices_twap - sell_prices_twap)
Percent Reduction		40.88983051	
INTC	Buy	0.005087719	np.mean(twap_prices - exec_prices)
INTC	Sell	0.005614035	np.mean(exec_prices - twap_prices)
Mean Difference (Algo)		-0.00070175	np.mean(buy_prices_algo - sell_prices_algo)
Mean Difference (TWAP)		0.01	np.mean(buy_prices_twap - sell_prices_twap)
Percent Reduction		107.0175	

Results Analysis

It’s visibly noticeable that our algorithm mostly performs better or same as TWAP!!!

Let’s analyze the results of both IS and OS:
Our algorithm *performed best on INTC and MSFT* in both the in-sample (IS) and out-of-sample (OS) periods, and *performed worst on AAPL and AMZN.* ***This outcome is largely driven by the nature of our weighted signal framework—volume imbalance is typically assigned a high weight, and its effectiveness heavily depends on the depth and liquidity of the order book***.

INTC and MSFT consistently exhibit higher bid-ask volumes compared to the other stocks, which amplifies the impact of the volume imbalance signal and leads to stronger execution performance.

Concluding Remarks

First, our results demonstrate that using weighted signals combined with time-based heuristics significantly improves execution performance in both in-sample and out-of-sample periods. This validates the importance of intelligently prioritizing signals based on their predictive strength.

Second, we showed that dynamic thresholds, such as those based on rolling quantiles, are essential for adapting to changing market conditions—especially in the high-frequency domain where static thresholds often fail to capture microstructure shifts.

Finally, we highlighted the necessity of using multiple features to build a more generalized algorithm, as different stocks exhibit distinct behaviors depending on their volume profiles and market demand.

One surprising takeaway was the insight that low-cost stocks often trade with significantly higher volume, a structural pattern we had not anticipated but which clearly impacted our signal effectiveness.