ORIE 5259
Market Microstructure and Algorithmic Trading: *Final Report*

Rethyam Gupta, Adarsh Pandey, Atharwa Pandey

April 2025

Abstract

In this report, we provide a summary of our work throughout the semester, focusing on the development and evaluation of execution algorithms for both the buy and sell sides. The core objective is to minimize the difference between the average buy and sell prices—effectively, to buy low and sell high. On the buy side, we aim to act before favorable price movements fully materialize, avoiding the cost of being late to a rising market. On the sell side, the goal is to exit positions before adverse price moves occur, capturing value while liquidity remains favorable. Our strategy emphasizes adaptability over static rules, leveraging rolling quantiles to adjust to evolving market conditions in real time.

Key variables integrated into the algorithm include bid-ask spread tightness, volume imbalance, trade intensity anomalies, short-term momentum shifts, deviations from recent volume curves, and an aggression ratio capturing the frequency of aggressive trades. These signals were selected for their sensitivity to short-term market dynamics and their ability to flag opportunities for favorable execution.

We detail the reasoning behind each signal derived from the variables mentioned above and describe our process for optimizing the strategy for individual stocks. This includes introducing both signal-specific weights and time-based dependencies, allowing the algorithm to dynamically prioritize signals that have historically improved execution quality in a given context. We also explain how these individual signals were ultimately integrated into cohesive decision-making frameworks for both the buy and sell sides. Our empirical analysis on AAPL and AMZN evaluates signal relevance, strategy performance, and robustness, and we benchmark our adaptive algorithm against a TWAP baseline (buying/selling at the first available option in each minute) to assess performance.

1. Strategy Overview and Key Findings

The core intuition behind the strategy is straightforward: we aim to buy before it becomes too late—when the price has already started moving up—and sell before the price begins to decline. To achieve this, we use a combination of market signals that capture liquidity conditions, short-term momentum, and order flow pressure. These include bid-ask spread tightness, volume imbalance, trade intensity, short-term momentum, deviations from recent volume curves, and an aggression ratio. These variables and their conversion into actionable signals are discussed in detail in the sections ahead.

Instead of relying on static, rule-based thresholds, we adopt an adaptive framework where each signal is dynamically compared against a rolling quantile-based threshold. If the aggregated signal exceeds a predefined execution threshold, the algorithm triggers a trade. This approach enables the strategy to remain responsive to changing market conditions and reduces the risk of overfitting to a single regime.

Mathematically, let the set of signals be denoted by $\{s_1, s_2, \ldots, s_n\}$, where each s_i corresponds to a real-time indicator such as spread tightness or volume imbalance. We assign a weight w_i to each signal based on its historical effectiveness in improving execution quality for a given stock. The aggregated weighted signal score is then given by:

$$\text{Trigger Action if} \quad \sum_{i=1}^{N} w_i \, s_i > T \tag{1}$$

The weights $\{w_i\}$ and the threshold T—a heuristic value that varies across time zones within a minute—are optimized per stock using backtesting on historical high-frequency data, with adjustments based on recent performance over a rolling window. This design ensures that the algorithm adapts to stock-specific behaviors and evolving market dynamics.

Constructive Aspects: One key advantage of using a weighted approach in our strategy is that it enables the algorithm to emphasize signals that are particularly informative for certain stocks, while down-weighting less relevant indicators. *For example, volume imbalance proves to be a stronger signal for stocks like MSFT and INTC, whereas bid-ask spread tightness plays a more significant role in stocks like GOOGL and AAPL.* This stock-specific characterization becomes more evident in later sections, where we discuss the signal weights assigned through empirical evaluation and strategy implementation.

Furthermore, the execution threshold T is not static—it varies across sub-intervals within a given minute. For instance, if trades executed in the first 15 seconds of a minute exhibit lower positive impact on the final price compared to those in the subsequent 15–30 second window, the threshold T during the initial period can be set higher to discourage earlier execution. This time-sensitive adjustment allows the strategy to capture favorable trading opportunities that might otherwise be missed under a uniform threshold, thereby improving overall execution quality. To support adaptiveness, each signal is computed relative to a rolling quantile over a recent time window. This rolling quantile ensures that the thresholds used to determine whether a signal is "active" adjust to the prevailing market regime.

Limiting Aspects: While the weighted signal-based strategy offers adaptability and stock-specific customization, it is not without limitations. First, identifying optimal weights and time-based thresholds is a non-trivial task, especially given the highly dynamic and noisy nature of high-frequency financial data. The market structure evolves rapidly, and what may appear optimal in a backtest may quickly become suboptimal in live conditions. Even with well-tuned weights, and thresholds, the strategy can still fail to respond to sudden, unforeseen market movements—particularly if the underlying cause is not captured by any of the active signals or is insufficiently weighted. In such cases, valuable trading opportunities may be missed.

Second, it is important to note that the analysis presented in the sections ahead is conducted under the simplifying assumption of zero slippage and instantaneous calculation of signals. In reality, slippage—especially during periods of high volatility or low liquidity—can have a significant impact on execution performance.

1.1. Key Factors and Discussions

- *By assigning weights to each signal, the algorithm inherently places greater emphasis on those indicators that are more informative for a particular stock.* For instance, volume imbalance consistently contributed to better execution decisions for `MSFT` and `INTC`, while bid-ask spread tightness proved to be more relevant for tickers like `GOOGL` and `AAPL`. This selective amplification allows the strategy to remain focused and avoid noise from less meaningful signals.

- *One trade is executed per minute, but we incorporated time-zone specific thresholds within the minute to introduce temporal sensitivity.* For instance, a higher threshold was applied during the first 15 seconds and a lower one in the following 15-second window. This design choice—aimed at avoiding early and potentially suboptimal execution—was informed by the empirical analysis discussed in the sections ahead.

- *To maintain robustness across different market regimes, we introduced a rolling quantile framework for threshold calibration at the level of individual signals.* Rather than relying on static cutoffs, each signal's threshold adapts continuously based on recent data distributions, allowing the strategy to remain responsive to evolving microstructural conditions and maintain execution quality over time.

- Given the granularity of features such as second-level volume metrics, momentum, and order book spreads, our framework operates under the assumption of zero latency and no slippage. All derived statistics—including moving averages, quantiles, and volatility metrics—are assumed to be computed instantaneously and without execution lag.

  While this assumption simplifies backtesting, it is not realistic in a live trading environment and may overstate the strategy's real-world performance.

- While our strategy focuses on identifying optimal weights for signals and zone-wise thresholds, it is important to recognize the challenges introduced by increasing granularity in this search. *Finer resolution in weights or thresholds expands the number of variables significantly, increasing both the complexity and computational cost of optimization.* In the sections ahead, we detail the structure of our search space and outline how we avoided brute-force grid search by leveraging a combination of data-driven analysis and domain intuition to guide the parameter selection process.

- During testing, the strategy yielded the strongest performance on **INTC** and **MSFT**, both of which were characterized by a heavy weighting on volume imbalance signals. In contrast, performance was poorest on **AMZN** and **AAPL**, where bid-ask spread tightness dominated but did not offer the same predictive strength as volume imbalance did for `INTC` and `MSFT`. These results emphasize the asset-specific nature of signal effectiveness and highlight the need for adaptive weighting across different instruments.

2. Features/Variables and Feature Engineering

Before we delve into the signals and the execution algorithm, it is essential to first understand the raw features and the feature engineering used to construct the key market features that are later transformed into actionable signals.

In designing the execution algorithm, we focus on six key market features: *Spread, Volume Imbalance, Momentum, Volume Curve Deviation, Trade Intensity, and Aggression Ratio.* These variables were carefully selected and engineered to ensure that the strategy captures a comprehensive range of market microstructure dynamics, including order book pressure, short-term price movements, deviations from expected trading volume profiles, liquidity conditions, trade flow intensity, and the aggressiveness of market participants. The goal is to construct a well-rounded and responsive strategy that integrates diverse market signals, each contributing unique predictive value. In the discussion that follows, we describe each feature in detail, outline how it is computed or engineered, and provide the intuition behind its inclusion in the strategy.

2.1. Spread

The bid-ask spread, or simply the spread, measures market liquidity. A narrower spread typically signals a more liquid and competitive market, whereas a wider spread can indicate market inefficiency or elevated risk.

The spread is calculated and used in the code as:

```
df['spread'] = df['ask price'] - df['bid price']
```

Incorporating the spread into the execution logic ensures that the algorithm avoids transacting during illiquid periods, which directly helps minimize slippage and trading costs (though our objective in execution going forward will not be to minimize slippage, this consideration helps make the strategy robust). Executing trades when the spread is tight allows us to secure better price fills, thereby improving the overall effectiveness of the strategy.

2.2. Volume Imbalance

Volume imbalance measures the relative strength between buyers and sellers based on order book depth. A large positive imbalance suggests dominant buying pressure (more bid volume than ask volume at the best prices), while a negative imbalance indicates selling pressure (more ask volume than bid volume at the best prices). We use volume imbalance because it provides a real-time snapshot of demand and supply forces in the market. Aligning executions with the prevailing imbalance helps the strategy move with the natural flow of the market, improving the probability of favorable trade outcomes.

It is theoretically defined as (also in code):

```
(df['bid volume']-df['ask volume'])/(df['bid volume']+df['ask volume'])
```

2.3. Price Momentum

Short-term price momentum captures the directional movement of price over a recent time window. Positive momentum signals ongoing upward pressure, whereas negative momentum indicates downward trends.

The momentum variable is computed as:

```
 df['momentum'] = df['ask price'].diff(periods=10)
```

This code calculates the change in the ask price over the past 10 ticks (for the buy side algorithm, it will use the bid price for the sell side).

Including momentum helps the algorithm align trades with short-term price trends rather than trading against them. We use momentum because, as we will discuss later, the goal is to enter the market early during an upward move for the buy side and exit before a downward move begins for the sell side. This improves the chances of achieving better execution prices and reduces the risk of adverse movements immediately after execution.

2.4. Volume Curve Deviation

Volume curve deviation helps identify abnormal volume behavior by comparing recent short-term volume patterns to their longer-term average. This signal captures deviations from the expected intraminute volume profile, allowing the strategy to detect moments of unusual market activity or hidden liquidity.

The Volume Curve Deviation is computed in the code as:

```
df["vol_curve_baseline"] = df["vol_curve_30s"].rolling("5min").mean()
df["vol_curve_deviation"] = df["vol_curve_30s"] - df["vol_curve_baseline"]
```

Here, `volume_curve_30s` represents the percentage of volume traded in the current 30-second interval, and the baseline is a 5-minute rolling average of this metric. A positive deviation suggests that the current volume is significantly higher than normal for that time segment, potentially indicating large participant activity or market inefficiencies.

2.5. Trade Intensity

Trade intensity captures the frequency of trades occurring within a short time window, serving as a proxy for market activity and urgency. A sudden increase in trade intensity often reflects heightened interest from market participants and may signal a short-term shift in demand or supply dynamics.

Trade Intensity is calculated in the code as:

```
df["trade_intensity"] = df["order id"].rolling('15s').count().values
```

In this formulation, each trade is assumed to be associated with a unique `order id`, and the count of trades over a 15-second rolling window is used to gauge real-time trading pressure. Higher trade intensity suggests increased market participation, often associated with stronger price moves or breakout conditions. By incorporating trade intensity into the execution logic, the algorithm is better positioned to time its actions during periods of active trading, where there may be more liquidity and lower market impact.

2.6. Aggression Ratio

Aggression ratio is designed to capture the degree of trading aggressiveness in the market by identifying instances where market participants are actively hitting bids or lifting offers. Such behavior often reflects strong conviction or urgency in trade direction and can signal imminent price movement.

The Aggression Ratio is computed in the code as:

```
df["delta_ask_vol"] = df["ask volume"].diff()
df["delta_bid_vol"] = df["bid volume"].diff()
df["aggressive_trades"] = ((df["delta_ask_vol"] < 0).astype(int) +
(df["delta_bid_vol"] < 0).astype(int))
df["aggression_ratio"] = df["aggressive_trades"].rolling('30s').mean()
```

This feature is based on the intuition that a sudden drop in either the bid or ask volume indicates that a market order has consumed available liquidity at that level—an aggressive act. By monitoring the proportion of such aggressive trades over a 30-second window, the aggression ratio quantifies the market's urgency and directional pressure.

Integrating the aggression ratio into our execution strategy allows the algorithm to react more intelligently to bursts of directional trading activity. A high aggression ratio suggests strong one-sided interest and can serve as a confirmation signal, especially when aligned with other features.

The use of this and other signals within the adaptive execution framework will be discussed in detail in the forthcoming sections.

3. Execution Strategy Design and Implementation

In this section, we explain the logic and workings of the execution algorithm for both the buy and sell side, along with the assumptions made during its implementation. We will then proceed to describe each signal in depth, detailing how they are constructed and interpreted.

The execution strategy is built around dynamically identifying favorable moments to enter and exit trades based on real-time market conditions. *Buy logic focuses on capturing early signs of market strength to enter the market, while Sell logic is designed to detect emerging market weakness to enable timely exits.* One important aspect of the execution algorithm is that it continuously looks for signal confirmations during the first 59 seconds of each minute. If the conditions are not satisfied within that time frame, it defaults to executing at the first available opportunity.

The core principle of the execution algorithm is multi-signal confirmation:

- A trade (either Buy or Sell) is executed only when the weighted combination of signals exceeds a time-zone-specific threshold T, as introduced earlier and further illustrated in the flowchart ahead. *This ensures that decisions are not driven by a single noisy indicator but are instead supported by a meaningful aggregation of multiple market features, calibrated based on the time of execution.*

The basic motivation for this design is twofold:

- To strike a balance between reactivity and reliability—triggering trades based on a single signal can lead to false positives during noisy or directionless market phases, while requiring all signals to align would result in missed opportunities during fast-moving conditions.

- To emphasize signals that are more informative for individual stocks by assigning them higher weights, while simultaneously avoiding suboptimal execution through the use of time-zone-specific thresholds T that delay trades when early signals are weak.

Additionally, the algorithm uses adaptive, dynamic thresholds instead of fixed cutoffs:

- Each indicator's threshold is calculated based on rolling quantiles over a short historical window (e.g., past five minutes). *This ensures the strategy stays sensitive to real-time changes in market structure — such as volatility, liquidity, spread, or momentum.* (Discussed in detail near the end of this section while summarizing the execution)

3.1. Implementing Signals: Explaining Signals and Logic

We now expand on how each of the previously discussed variables is converted into a signal. For each signal, we also provide the reasoning behind its construction for both the buy and sell sides. In this section, we focus primarily on the logic used to activate the signals. A brief discussion at the end of the section outlines how we arrived at the specific quantile thresholds—such as using the 10th percentile `(q10)` for Spread and other similar choices—based on empirical observations.

3.1.1. Bid-Ask Spread Tightness: The signal is triggered when the current spread falls below the 10th percentile of recent spreads for both Buy and Sell decisions.

```
Signal: spread < spread_q10
```

Reasoning: A tight bid-ask spread indicates high liquidity and low transaction costs, favorable conditions for both entering (Buy) and exiting (Sell) trades for our purpose. By conditioning trades on tight spreads, the strategy minimizes slippage and improves execution quality, especially during fast-moving markets when spreads can otherwise widen rapidly.

3.1.2. Volume Imbalance: The signal is triggered when the current volume imbalance exceeds the 95th percentile of recent imbalances (for Buy decisions), or falls below the 5th percentile (for Sell decisions). Recent means a five-minute lookback for adaptive thresholds.

```
Buy:  volume_imbalance > vol_imb_q95 || Sell: volume_imbalance < vol_imb_q05
```

Reasoning: Volume imbalance measures the dominance of bid-side or ask-side liquidity in the order book. A strong positive imbalance (Buy side) indicates substantial buying interest, while a strong negative imbalance (Sell side) signals selling pressure. Using rolling quantiles ensures that the threshold is adaptive to the market's evolving liquidity conditions, helping the strategy capture genuine shifts rather than reacting to noise.

3.1.3. Price Momentum: The signal is triggered when recent price momentum exceeds the 80th percentile (for Buy decisions) or falls below the 20th percentile (for Sell decisions).

```
Buy:  momentum > momentum_q80 || Sell: momentum < momentum_q20
```

Reasoning: Momentum captures the short-term direction and strength of price movement. Strong positive momentum suggests that prices are accelerating upwards (Buy side), while strong negative momentum indicates a downward acceleration (Sell side). By anchoring the signal to rolling quantiles, the system dynamically adjusts to different market volatility regimes, ensuring that only unusually strong momentum is acted upon.

3.1.4. Trade Intensity: The signal is triggered when the current trade intensity falls below the 10th percentile, for both Buy and Sell decisions.

```
Signal:  trade_intensity < trade_intensity_q10
```

Reasoning: Trade intensity reflects how frequently trades are occurring in the market within a short time window. Lower intensity is often observed just before major activity begins, making it a valuable cue for early entry (Buy side) or early exit (Sell side). By acting during quiet phases, the strategy positions itself ahead of the crowd, improving the chances of favorable execution without excessive competition.

3.1.5. Volume Curve Deviation: The signal is triggered when the volume traded in the current interval is significantly lower than expected—specifically, when the deviation falls below the 20th percentile. This applies to both Buy and Sell decisions.

```
Signal: volume_curve_deviation < volume_curve_deviation_q20
```

Reasoning: A negative deviation from the expected volume curve suggests that the current trading activity is quieter than usual for that point in time. This early-phase under-participation can serve as a valuable indicator for anticipating upcoming volume surges. Executing during this lull allows the strategy to enter or exit positions before the broader market becomes active, improving execution quality.

3.1.6. Aggression Ratio: The signal is triggered when the aggression ratio exceeds the 80th percentile, for both Buy and Sell decisions.

```
Signal: aggression_ratio > aggression_ratio_q80
```

Reasoning: A high aggression ratio indicates that market participants are actively lifting offers or hitting bids, reflecting urgency and directional conviction. This behavior often precedes short-term price moves, making it a strong indicator of momentum. By reacting to elevated aggression levels, the strategy can align itself with dominant order flow, improving the probability of participating in meaningful directional moves.

There are two important assumptions to know in the design of our execution algorithm. First, we assume no slippage, meaning that trades are executed immediately at the quoted bid or ask price without any price slippage. Second, we assume efficient quantile adaptation, where rolling quantile estimates adjust quickly enough to reflect recent market changes without introducing significant lag.

Both assumptions essentially imply that there is no slippage or time gap between signal generation and execution, but we believe that both aspects should be explicitly mentioned.

To Summarize: A trade order — either a Buy or a Sell — is executed when the weighted combination of six signals exceeds a time-zone-specific threshold T. This multi-signal framework was designed to strike a deliberate balance: avoiding both excessive reactivity to noise and overly conservative hesitation. Instead of relying on a single trigger, the algorithm acts only when a sufficiently strong combination of market signals suggests a genuine opportunity.

The six signals discussed in the previous sections each capture a distinct aspect of market microstructure — from directional pressure and short-term acceleration to order flow behavior, liquidity tightness, and deviations from typical volume profiles. Together, they form a diverse and complementary set of indicators, enabling the strategy to adapt across a variety of market conditions.

Moreover, all signal thresholds are adaptive, computed using quantile-based filters over short rolling windows (five minutes in our implementation). Rather than applying rigid, static cutoffs, this method allows the system to define “high” or “low” for each signal

based on recent data — ensuring relevance across changing volatility and liquidity regimes. This adaptive calibration is crucial for maintaining robustness without requiring manual intervention or constant parameter tuning.
Development Pathway: Our final framework is the result of several iterative refinements, made possible through internal experimentation and continuous feedback. We began with a simple, equally weighted signal model using a fixed rule (e.g., trigger if more than 3 signals were active). Through multiple team discussions — and with special thanks to Umu for his constructive insights during review meetings — we moved toward a more nuanced approach, introducing per-stock weighting to reflect signal importance. Further refinements included the integration of time-based thresholds, improving the system's robustness to premature or low-quality trades.

In addition to this conceptual development, we also support our final design through empirical validation. Right ahead, we present a flowchart that encapsulates our entire algorithm in a single image, specifically for the Buy-side logic. With simple substitutions in signal directions, the same structure can be extended to the Sell-side algorithm. Following this, we present plots that demonstrate the importance of both weighted signals and time-based heuristic thresholds. We also include plots that illustrate how the thresholds were selected, based on empirical observations, thereby substantiating the design choices made in the final execution logic.

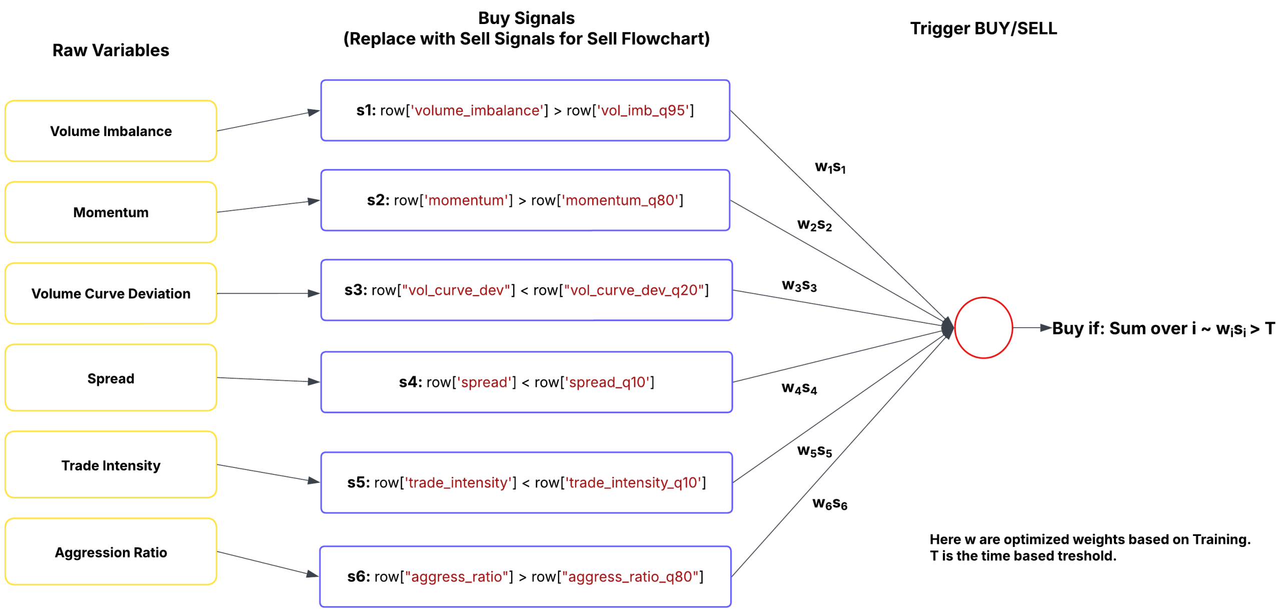

Raw Variables
Buy Signals
(Replace with Sell Signals for Sell Flowchart)
Trigger BUY/SELL
Volume Imbalance
Momentum
Volume Curve Deviation
Spread
Trade Intensity
Aggression Ratio
s1: row['volume_imbalance'] > row['vol_imb_q95']
s2: row['momentum'] > row['momentum_q80']
s3: row["vol_curve_dev"] < row["vol_curve_dev_q20"]
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.

Figure 1: The above figure provides a thorough overview of the entire execution algorithm — Buy in this case — but with simple replacements, it yields the Sell execution algorithm.

Up until Milestone 2, our algorithm adopted a simple rule: an action would be triggered if the number of active signals exceeded two — that is, if at least four out of six signals were triggered. Formally, let $s_1, s_2, s_3, s_4, s_5, s_6 \in \{0, 1\}$ denote the binary states of the four signals. The original decision rule was:

$$\text{Trigger Action if } \sum_{i=1}^{6} s_i > FixedTreshold \tag{2}$$

This approach treated all signals equally and ignored any temporal variation in execution sensitivity.

However, each signal displays a different frequency of activation as discussed in the Milestone 2 report. This disparity suggested that equal weighting may overlook key differences in predictive power. Hence, we transitioned to a weighted signal formulation, assigning weights w_i to each signal and comparing the weighted sum against a fixed threshold T:

$$\text{Trigger Action if} \quad \sum_{i=1}^{6} w_i \, s_i > FixedTreshold \tag{3}$$

While this allowed more reliable signals to exert greater influence, it still assumed a static decision threshold throughout the minute. To better capture evolving market conditions within each minute, we further enhanced the model by introducing time-based thresholds. Let $z \in \{0\text{–}15\text{s}, 15\text{–}30\text{s}, 30\text{–}45\text{s}, 45\text{–}59\text{s}\}$ denote the subinterval of the minute. Our final execution logic now triggers an action according to:

$$\text{Trigger Action if} \quad \sum_{i=1}^{4} w_i \, s_i > T_z \tag{4}$$

Here, T_z varies depending on the execution zone within the minute, allowing the strategy to defer or accelerate trades depending on market behavior in that subinterval.

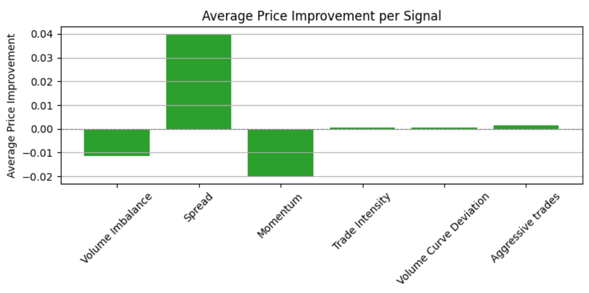


Figure 2: AAPL BUY

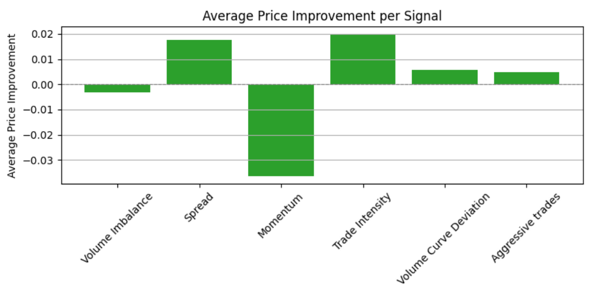


Figure 3: AMZN BUY

As discussed earlier, when signals are equally weighted, each contributes differently to the execution performance, resulting in significant variability in average price improvements. In both Apple and Amazon, some signals display a consistent positive impact while others contribute negatively. This discrepancy highlights the need for a weighted strategy, where signals with negative impacts are down-weighted or excluded, while those with positive impacts are emphasized. The weighted approach dynamically adjusts signal influence, aligning the execution strategy more closely with historical performance.

Furthermore, the motivation behind incorporating time-based thresholds lies in optimizing the tradeoff between when a trade is executed and the resulting price improvement. The goal is not to ensure an even distribution of trades across the minute, but rather to prioritize execution when the expected improvement is favorable. For instance, in Apple's case (Figures 4 and 5), we observe consistently poor price improvement during the early segments of each minute despite a high execution count, indicating that trades in those intervals should be discouraged through stricter thresholds. In contrast, later intervals (e.g., 45–59s) offer better improvements and can be prioritized. *The plots shown above and below correspond to the training period before optimization, where a fixed threshold of 3 was used across all time zones and equal weights were assigned to all signals.* These observations form the basis for transitioning to a time-aware, performance-driven execution strategy.

Before we conclude this section and proceed to the training and performance evaluation phase, we wanted to include an important detail we had initially planned to add at the end

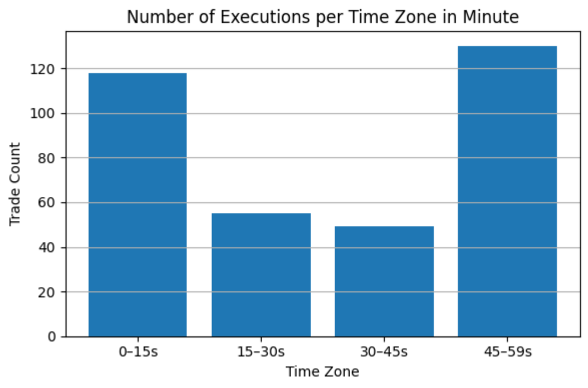


Figure 4: AAPL EXECUTION TIME

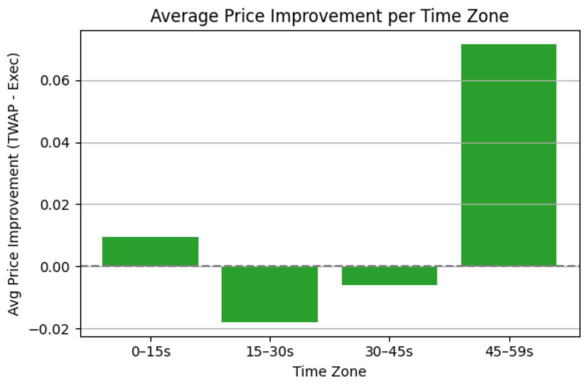


Figure 5: AAPL PRICE IMPROVEMENT

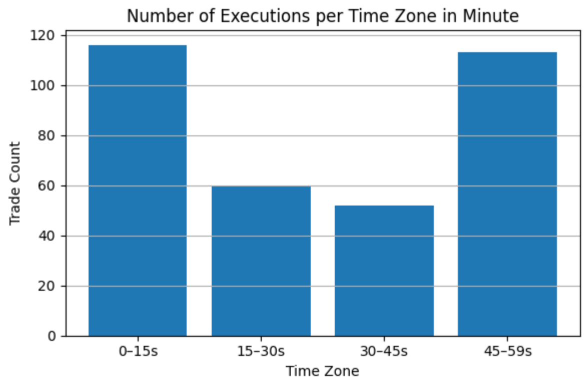


Figure 6: AMZN EXECUTION TIME

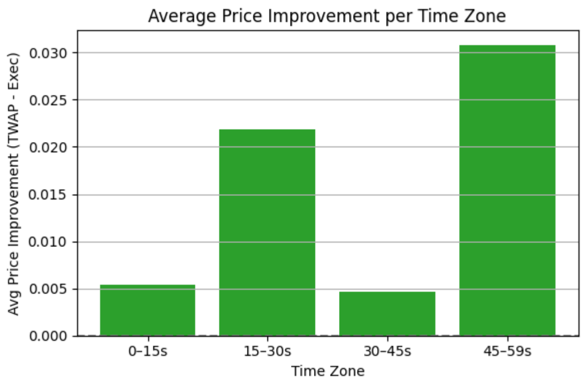
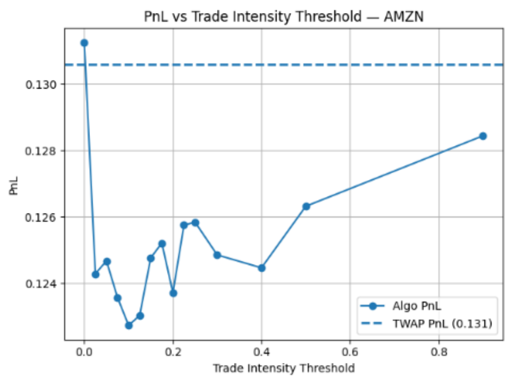
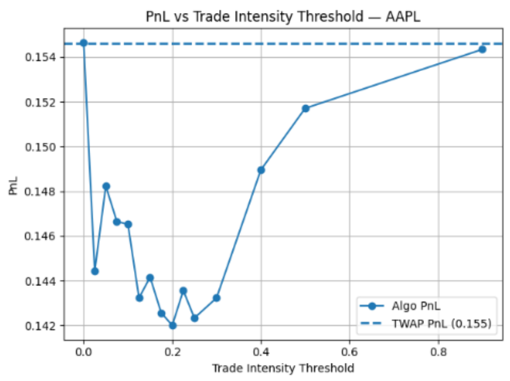


Figure 7: AMZN RICE IMPROVEMENT

of this section—namely, how we selected the thresholds used in our rolling quantile strategy. For example, as illustrated in the Figure below, we conducted a threshold sweep over various values of the trade intensity signal and plotted the corresponding PnL curves for both AMZN and AAPL. Based on the observed turning points and stability in these plots, we chose a threshold that balanced performance and robustness. This technique was inspired by a methodology briefly introduced in one of our lectures, and we found it sufficiently effective to incorporate into our project pipeline as well.





4. Training and Performance Analysis

Thus far, we have discussed nearly every component of our execution framework except for its training and performance. We began by introducing the core variables and market features in detail, followed by an explanation of how these evolved into meaningful signals. We then examined the decision algorithm from the ground up, outlining its logic, structure, and the iterative improvements that led to the current version — including weighted signals and time-sensitive thresholds. Having established the complete execution logic, we now turn to the final stages of our study: how the algorithm was trained and an evaluation of its performance based on empirical results.

4.1. Training

Since financial data evolves over time and exhibits non-stationary behavior, the most appropriate approach for training and validation in such contexts is Walk-Forward Validation. In this technique, the model is trained on a rolling historical window and validated on the immediate next time block. The window then shifts forward, simulating real-time trading conditions and preserving temporal integrity. This sequential retraining process helps avoid look-ahead bias and ensures that performance metrics reflect realistic, out-of-sample behavior. In our framework, the key parameters to be optimized were the weights assigned to each signal. These weights determine how much influence each signal has in the final decision rule. Our objective was to minimize the mean (buy price - sell price).

It is important to note that while signal weights were optimized quantitatively, the time-based threshold T_z was not analytically optimized. Instead, we relied on visual inspection of empirical performance plots to make intuitive adjustments. This approach allowed us to maintain simplicity while still capturing intraminute dynamics effectively.

To perform the optimization efficiently, we avoided exhaustive *grid search* due to its high computational cost. Instead, we employed the Optuna library — a state-of-the-art hyperparameter optimization framework that uses Bayesian optimization and Tree-structured Parzen Estimators to explore the search space intelligently. Optuna allowed us to focus on promising regions of the weight space quickly, improving convergence and scalability.

It is worth emphasizing that, due to computational and time limitations, we restricted the weight values to integers in the range $[0, 4]$ for each signal. While this limited space was sufficient to observe meaningful performance differences, increasing the granularity — for example, allowing continuous weights or extending the range — would almost certainly yield even better results.

To provide a good flow of our optimization, the process followed an iterative design under a walk-forward validation framework. The steps were as follows:

1. Initial Weight Optimization (Fixed Thresholds): We began by optimizing the signal weights using fixed time thresholds across all execution intervals. The objective was to maximize the difference between average sell and buy prices.

2. Threshold Zone Analysis: With the initial weights in place, we examined price improvement plots across different sub-minute intervals (0–15s, 15–30s, etc.) to identify regions where execution impact was stronger or weaker.

3. Time-Based Threshold Allocation: Based on the insights from these empirical plots, we assigned updated time-sensitive thresholds T_z, adjusting the execution aggressiveness depending on the observed market behavior within each zone.

4. Final Re-optimization: With time-based thresholds now introduced, we re-optimized the signal weights again — this time within the context of the new thresholds — to arrive at the final configuration of weights and thresholds used in our execution logic.

This entire process was implemented using Walk-Forward Validation, where the data was split into sequential rolling windows. Each window consisted of 30 minutes of training data followed by 15 minutes of validation. After each iteration, the training window was expanded to include the previous validation set, mimicking a realistic setting where models continuously learn from newly available data. This ensured robustness against overfitting and maintained forward-looking integrity throughout the development.

Ticker	**Buy/Sell**	**0–15s**	**15–30s**	**30–45s**	**45–59s**	**w0**	**w1**	**w2**	**w3**	**w4**	**w5**
GOOG	Buy	5	3	3	3	0	4	1	0	0	0
GOOG	Sell	3	3	3	3	3	4	0	1	1	0
AAPL	Buy	4	4	4	3	1	4	2	1	0	0
AAPL	Sell	5	3	3	3	1	4	0	0	0	2
MSFT	Buy	4	4	3	3	4	2	0	0	0	0
MSFT	Sell	4	4	3	3	3	1	1	0	0	2
AMZN	Buy	5	4	3	3	2	4	2	1	0	0
AMZN	Sell	5	4	3	3	2	4	3	0	0	2
INTC	Buy	5	3	3	3	4	2	0	0	0	0
INTC	Sell	5	3	3	3	4	2	0	0	1	2

Table 1: Time-based thresholds and optimized weights for each ticker and execution side.

Interpretation: The weights w_0 through w_5 correspond to the following six signals respectively: `[volume_imb, spread, momentum, trade_intensity, volume_curve_dev, aggression]`. From Table 1, it is evident that different stocks prioritize different signals. For instance, both INTC and MSFT exhibit a strong preference for `volume imbalance` (w_0), suggesting that order flow asymmetry plays a significant role in their execution dynamics. On the other hand, stocks like GOOG, AAPL, and AMZN assign relatively higher weights to `spread` (w_1), highlighting the importance of liquidity tightness in influencing trading decisions. This diversity in signal weighting underscores the adaptive nature of the strategy and the need to tailor signal influence based on stock-specific microstructure behavior.

It is also quite evident that for nearly all stocks, the threshold in the first 15-second window (0–15s) is higher than in subsequent intervals. This can be attributed to the fact that trades executed in the very beginning of a minute tend to have a higher level of uncertainty and market noise. By requiring a stricter threshold early on, the algorithm avoids prematurely entering trades based on volatile or weak signals, thereby improving the precision of execution timing.

4.2. Performance Analysis

Before evaluating the performance of our execution algorithm, it is important to first define the benchmark we are comparing against. In our case, the benchmark is the TWAP strategy. To shed more light on this, TWAP simply buys or sells at the very first available opportunity without waiting for any market signal confirmation, whereas our execution algorithm deliberately waits for favorable signals before executing a trade.

To evaluate our algorithm against TWAP, we focus on whether we are able to achieve a lower average buy price, a higher average sell price, or, more simply, a lower difference between the average buy and sell prices compared to TWAP. If our algorithm consistently delivers a better price differential than TWAP, it provides strong justification to continue developing and refining the approach.

In addition to this, on top of the comparison, we also conduct a deeper analysis of the minute-by-minute execution behavior. Specifically, we examine how many trades our algorithm executes better than TWAP on both the buy and sell sides. Furthermore, we analyze the timing of executions within each minute: while TWAP always executes at the first available opportunity, our algorithm waits for signal confirmation, and it is important to study when and how often these decisions lead to better execution outcomes.

Table 2 presents a detailed comparison of our execution algorithm against the TWAP benchmark during both the training and testing periods. The table reports average improvements on the buy and sell sides, along with the mean difference between buy and sell prices and the corresponding percent reduction relative to TWAP.

The results clearly demonstrate that our algorithm consistently outperformed the TWAP strategy across both periods. While some performance degradation was observed in the testing phase—an expected outcome given the non-stationary nature of financial markets—the drop was not significant. In fact, in certain cases, such as **INTC**, the algorithm exhibited improved performance in the test period, managing to generate net profit.

Ticker	**Execution**	**Train Improv.**	**Test Execution**	**Test Improv.**
GOOG	Buy	0.084179	Buy	0.063181818
	Sell	0.05268	Sell	0.057954545
	Mean Diff (Algo)	**0.1373487**	**Mean Diff (Algo)**	**0.06840909**
	Mean Diff (TWAP)	0.27420749	Mean Diff (TWAP)	0.18954545
	Percent Reduction	**49.91066801**	**Percent Reduction**	**63.90887252**
AAPL	Buy	0.050597	Buy	0.043076923
	Sell	0.051932	Sell	-0.007179487
	Mean Diff (Algo)	**0.05207386**	**Mean Diff (Algo)**	**0.05871795**
	Mean Diff (TWAP)	0.15460227	Mean Diff (TWAP)	0.09461538
	Percent Reduction	**66.31753208**	**Percent Reduction**	**37.94037502**
MSFT	Buy	0.006293	Buy	0.003488372
	Sell	0.004137	Sell	0.004186047
	Mean Diff (Algo)	**-0.00028736**	**Mean Diff (Algo)**	**0.00255814**
	Mean Diff (TWAP)	0.01014368	Mean Diff (TWAP)	0.01023256
	Percent Reduction	**102.8328969**	**Percent Reduction**	**75**
AMZN	Buy	0.037595	Buy	0.0098
	Sell	0.026275	Sell	0.0288
	Mean Diff (Algo)	**0.06671554**	**Mean Diff (Algo)**	**0.0558**
	Mean Diff (TWAP)	0.13058651	Mean Diff (TWAP)	0.0944
	Percent Reduction	**48.91084845**	**Percent Reduction**	**40.88983051**
INTC	Buy	0.0047	Buy	0.005087719
	Sell	0.003682	Sell	0.005614035
	Mean Diff (Algo)	**0.00166177**	**Mean Diff (Algo)**	**-0.00070175**
	Mean Diff (TWAP)	0.01017964	Mean Diff (TWAP)	0.01
	Percent Reduction	**84.11761123**	**Percent Reduction**	**107.0175**

Table 2: Training vs. Testing improvements, mean difference comparisons, and percent reduction for each ticker.

It is important to emphasize that we were able to achieve close to 50% improvement in execution performance over the TWAP baseline in almost every case, with even higher improvements in certain cases. This substantial gain can be attributed to the use of an informed, signal-aware algorithm that assigns stock-specific weights to each signal based on its relative importance. As discussed extensively in earlier sections, different stocks respond more strongly to different market microstructure signals — such as volume imbalance, spread, or momentum. By identifying and emphasizing these signals through optimized weighting, our algorithm was able to make significantly more efficient execution decisions compared to TWAP, which lacks any market context or adaptability.

Moving ahead, we now narrow our focus to two representative stocks — AAPL and AMZN — to conduct a deeper analysis of our execution strategy's behavior, the reason being that they were the worst-performing ones compared to INTC and MSFT. By examining these cases in detail, we aim to understand how the optimized strategy performed during both the training and testing periods. Specifically, we compare instances where the algorithm outperformed, matched, or underperformed relative to TWAP. This closer inspection will help us uncover patterns behind the algorithm's decision-making and the market conditions under which it tends to succeed or struggle.

Before we discuss the upcoming figures, to briefly recap, our execution algorithm searches for favorable signals during the first 59 seconds of each minute, as previously discussed in detail. If no signal condition is satisfied within that window, the algorithm defaults to executing at the first available opportunity — namely, the last second. Based on this design, we can categorize the timing of our trades into three distinct paradigms:

- Trades executed at the same time as TWAP: indicating that our algorithm and TWAP execute simultaneously.
- Trades executed between the TWAP point and the last second: reflecting trades triggered by our signal conditions within the minute.
- Trades executed at the last second: indicating that no signal was triggered, and a forced execution occurred at the end of the window.

Trading at the same time as TWAP generally results in obtaining the same execution price as TWAP, offering no advantage. Trading between TWAP and the last second represents the ideal behavior we seek to maximize — where trades are initiated based on meaningful signal confirmation rather than arbitrary timing. Finally, trades executed at the last second are typically suboptimal, as they indicate that no signal was triggered in time, and the algorithm was forced to execute without informational advantage.

Out of these three categories, it is clear that the first (same as TWAP) and the third (last-second trades) are undesirable outcomes for us. The second category — trades between TWAP and the last second — is the one we aim to maximize, as it reflects trades influenced by actual market signals. However, it is important to note that even this category requires further improvement, because trading between TWAP and the last second does not automatically guarantee a better price or a profitable outcome. The quality of the signals remains crucial.

From the figure on the next page, it is evident that our algorithm consistently outperforms TWAP on the training data, as seen by the significant number of trades where our algorithm executes at better prices than TWAP—particularly for both AAPL and AMZN. However, this performance advantage narrows during the backtest (testing) phase. For the two worst-performing stocks—AAPL and AMZN—the number of trades where TWAP was better is nearly equal to the number where our algorithm was better. This convergence helps explain their relatively poorer performance compared to other stocks like INTC and MSFT.

Upon closer inspection, it can be observed that AAPL and AMZN were also subject to high execution thresholds in the first 15 seconds of each minute. Both strategies heavily relied on the spread signal to make trading decisions. Therefore, due to stringent early thresholds, our algorithm may have missed opportunities that were more favorable in the later time zones. In contrast, for other stocks, our algorithm achieved significant improvement—up to 80% better executions overall, and a remarkable 107% improvement with INTC—where such restrictive thresholds or dependencies were not present. This observation supports the need for dynamic threshold calibration tailored to each stock's microstructure characteristics.

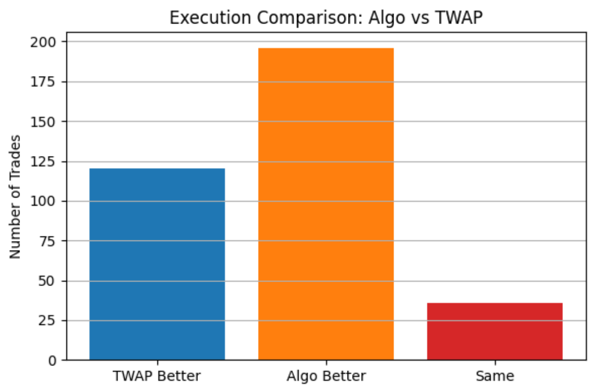


Figure 8: AAPL Train data

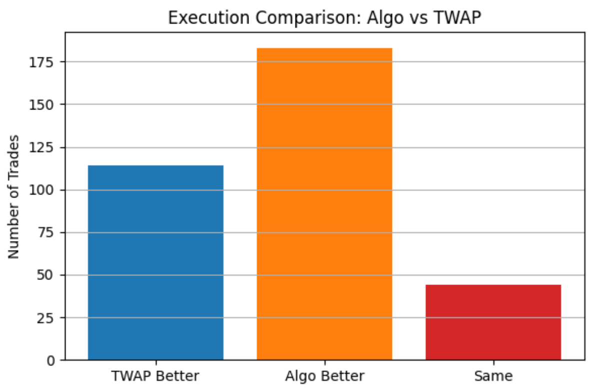


Figure 9: AMZN Train data

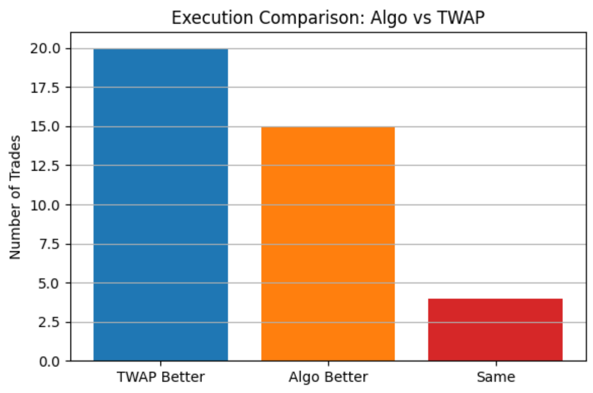


Figure 10: AAPL Backtest data

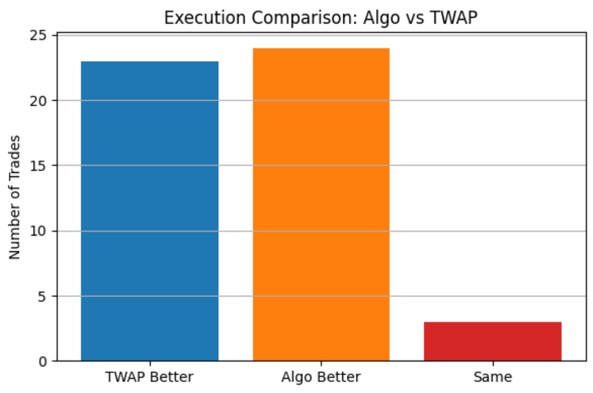


Figure 11: AMZN Backtest data

We have intentionally focused on the two worst-performing stocks in this analysis to emphasize both the strengths and, more importantly, the weaknesses of our approach. By highlighting what went wrong, we aim to extract lessons that inform future improvements. We assure the reader that results for other stocks—including INTC and MSFT—are significantly better, as can also be verified from the main summary table in the report. The purpose here is not to showcase success, but to reflect critically on failure where it occurred.

With this, our final report comes to an end. We would like to sincerely thank **Umu** for his insightful feedback and guidance throughout all stages of the project. We also extend our heartfelt gratitude to **Professor Stoikov** and **Professor Nuti** for their valuable comments and support.

This report contains all the essential tables and figures necessary for understanding our methodology and findings; hence, no appendix has been included. Any supplementary plots can be accessed through the submitted codebase and project presentation.