High-Frequency Execution Strategy Based on Market Microstructure Features

Kexin Deng(kd537), Yichen Gao(yg635), Dora Xu(dx33), Michael Cao(yc849)

May 18, 2025

Abstract

In this project, we develop and test a high-frequency trading algorithm that systematically executes buy and sell decisions based on real-time order book dynamics. Our objective is to design an execution strategy that intelligently responds to short-term market conditions to achieve superior trade prices relative to the prevailing market spread.

Trading decisions are guided by six key microstructure-based features: short-term momentum across various time windows, volatility-to-spread ratio, order book imbalance, order flow imbalance over a 10-second window, bid-ask spread, and time pressure. At each event, the strategy computes real-time buy and sell scores using a weighted linear combination of these features, ensuring that decisions reflect both liquidity conditions and directional signals.

Exactly one buy and one sell are executed per minute, with fallback rules ensuring timely action if no strong signals arise. Parameters are tuned in training period by minimizing the average (Buy Price – Sell Price), best parameter's results are compared relative to a TWAP benchmark in terms of average and standard deviation. Overall, this project demonstrates how integrating diverse, real-time microstructure signals through systematic scoring can improve execution timing and trading performance in high-frequency environments.

1 Key Messages

- **Strategy Overview**: The strategy dynamically adjusts to momentum, volatility-to-spread ratio, order flow, order book imbalance, bid-ask spread, and time pressure. It enforces exactly one buy and one sell per minute, and its effectiveness is measured by how much it reduces the spread compared to the TWAP benchmark.
- **Main Insight - Adaptive Scoring with Rolling Validation**: Using a rolling window approach, the strategy calculates a weighted score for each trade. A trade is only executed if it's the first buy or sell in the minute to surpass its respective threshold. The score weights are tuned through 500 randomized simulations to enhance performance.
- **Great Performance**: Out strategy was trained on one dataset and tested on separate backtest data. It significantly outperformed the TWAP benchmark in terms of spread reduction, achieving improvements of 39.71% (AAPL), 49.31% (AMZN), 70.60% (GOOG), 130.90% (INTC), and 126.19% (MSFT).
- **Key Strength - Comprehensive Fallback Logic, Rolling Windows Design and Early Execution Advantage**: A fallback rule ensures exactly one buy and one sell occur each minute, reducing variance during weak signal periods and improving execution reliability. The rolling window structure reflects real intraday market behavior by using only recent data for decision-making. Additionally, the strategy tends to execute trades early in each minute, helping it avoid rushed trades and outperform passive benchmarks.
- **Weaknesses**: While the rolling window and real-time feature updates make the strategy more realistic, they also introduce considerable computational load—especially during long backtests or when using more sophisticated weight schemes. Also, the tendency to trade early in the minute can result in missed opportunities that arise later. These issues could be addressed by integrating more adaptive, automatically-tuned thresholds.

2 Variable Construction

2.1 Feature Definition

1. **Momentum Factors**: Capture mid-price changes over different short-term horizons. Parameter in scoring function are

   - **Momentum 1**: Measures the mid-price change over a very short window (e.g., 1 s):

     $$\text{Momentum\_1} = \text{Mid\_price}_t - \text{Mid\_price}_{t-1} \tag{1}$$

     Indicates the most immediate price movement direction and strength.

   - **Momentum 3**: Measures mid-price change over a slightly longer window (e.g., 3 seconds):

     $$\text{Momentum\_3} = \text{Mid\_price}_t - \text{Mid\_price}_{t-3} \tag{2}$$

     Captures short-term trend development beyond immediate fluctuations.

   - **Momentum 5**: Measures mid-price change over an even longer window (e.g., 5 seconds):

     $$\text{Momentum\_5} = \text{Mid\_price}_t - \text{Mid\_price}_{t-5} \tag{3}$$

     Useful for detecting more stable micro-trends.

   Assets with strong recent performance tend to continue performing well in the short term. By giving the momentum factor positive weights in both score functions, the model is designed to favor assets exhibiting upward price trends in the buy-side algorithm, while favoring downward price trends in the sell-side algorithm.

2. **Volatility-to-Spread Ratio**: Evaluates short-term price volatility relative to the market spread:

   $$\text{Vol\_Spread\_Ratio} = \frac{\sigma(\text{Mid Price})}{\text{Ask Price} - \text{Bid Price}} \tag{4}$$

   where $\sigma(\text{Mid\_price})$ is the rolling standard deviation of mid-price. A higher ratio suggests higher volatility compared to liquidity, implying greater opportunity for execution. On the other hand, a low ratio indicates that the market may be stable or that the spread is wide, producing a less efficient and more illiquid market. This factor has a positive weight for buying and a negative weight for selling, since it is best to be active when liquidity is high and spreads are low.

3. **Order Flow (10s)**: Measures net aggressive trading at the best bid and ask prices over a time window of 10 seconds:

   $$\text{OFI}_{10}(t) = \sum_{\tau=t-9}^{t} (\text{Bid Volume} - \text{Ask Volume}) \tag{5}$$

   By analyzing the change over 10 seconds rather than tick by tick, we achieve smoother movement with reduced noise. A positive value indicates more aggressive buying which implies buying pressure from a potential upward price movement. Negative values imply more aggressive selling with downward price pressure. A larger order flow conveys a positive buy signal, while a smaller order flow indicates a sell signal; therefore, we assign a positive weight to this factor in both score functions.

4. **Order Book Imbalance**: Represents the direction of price movement based on bid and ask volume:

   $$\text{OBI} = \frac{\text{Bid Volume - Ask Volume}}{\text{Bid Volume} + \text{Ask Volume}} \tag{6}$$

   When imbalance is positive, there is buy-side pressure because there are more buy orders sitting on the bid side than sell orders on the ask side. A high positive imbalance can be interpreted as a bullish sign as it indicates price being pushed upward. On the other hand, a negative imbalance tends to result in downward price pressure as heavy sell-side liquidity can lead to price weakening. Similarly to order flow, we assign a positive weight to this factor in both score functions.

5. **Time Pressure**: Represents the fraction of the current minute that has elapsed:

$$\text{Time\_Pressure} = \frac{\text{Seconds\_Elapsed\_In\_Minute}}{60} \tag{7}$$

   Captures how much time has elapsed within the minute. The real-time score should rise with time pressure because regardless of signal strength, urgency increases to ensure that a required trade occurs. To facilitate trading when the remaining time is limited, we assign a positive weight to this factor in the buy-side score function and a negative weight in the sell-side score function.

6. **Bid-Ask Spread**: Represents the difference between the best ask price and the best bid price:

$$\text{Spread} = \text{Ask Price - Bid Price} \tag{8}$$

   Since the bid represents demand and the ask represents supply, the spread represents the transaction cost for immediate liquidity. A narrow spread suggests high liquidity, efficient markets, and low transaction costs while a wide spread indicates low liquidity and higher transaction costs for immediate execution. Because a wider spread is disadvantageous for both buying and selling due to higher slippage, this factor has a negative weight for buying and a positive weight for selling.

2.2 Parameter setting and Weight Simulation

To identify effective feature combinations, we simulate 500 random weight vectors for both the buy and sell scoring functions using a uniform sampling approach. The function `sample_weight_continuous()` generates 7-dimensional weight vectors corresponding to the seven standardized features. Positive weights (sampled from $[0, 1]$) are assigned to momentum, order flow, and imbalance features, encouraging directional execution. Negative weights (sampled from $[-1, 0]$) are used for volatility-to-spread ratio and spread to penalize trades in unfavorable liquidity conditions. For each buy-sell pair, the strategy is backtested, and the pair yielding the best average spread improvement is selected as the optimal weight configuration. The real-time buy and sell scores are computed using a linear combination of standardized features weighted by a parameter vector. Each feature's contribution is scaled by its empirical standard deviation to ensure comparability. The buy-side score function includes a constant coefficient of 0.5 on the `Time_Pressure` term, promoting early execution as time progresses. For the sell-side score, the same time pressure coefficient is subtracted, increasing sell urgency toward the end of each minute. Feature-specific directional adjustments are applied during sell scoring—for example, the weights for `Volatility-to-Spread Ratio` and `Spread` are negated to reflect their inverse impact on sell-side execution quality. The full set of weights is optimized during training and reused during backtesting for execution.

2.3 Feature Engineering

To reduce noise and improve signal robustness, we applied several feature engineering techniques. First, **rolling window averages** were used, such as computing rolling volume over a 20-tick window to smooth short-term fluctuations. We also derived custom microstructure features including the **bid-ask spread** (difference between best ask and bid prices) and **signed volume** (order size multiplied by direction). Each feature used in the scoring function was **standardized** by its rolling standard deviation (z-score normalization) to ensure scale comparability and allow consistent weighting. Furthermore, we extracted a fine-grained **time pressure** variable—measured as seconds elapsed within each minute—to model execution urgency as time progresses. These engineered features form the foundation of our real-time execution signal.

3 Strategic Logic

3.1 Feature Computation

Regarding feature computation, the algorithm computes a set of features for our limit order book data. These include multiple short-term momentum factors, the volatility-to-spread ratio, 10s order flow, the spread, the imbalance, and a time pressure variable reflecting the elapsed fraction of the trading minute. All computations rely solely on information observable at or before the current timestamp without reliance on future data.

3.2 Score Calculation

Using separate sets of weights for buying and selling, the module calculates a linear combination of the observed features to generate real-time scores. For each new event, a buy score and a sell score are calculated independently.

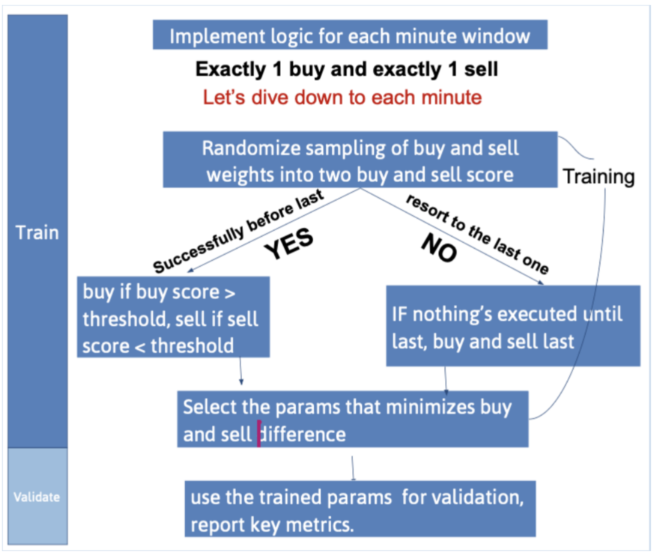
$$\begin{aligned}
\text{Buy\_Score} = {} & w_{1,\text{buy}} \times \text{Momentum1} + w_{2,\text{buy}} \times \text{Momentum3} + w_{3,\text{buy}} \times \text{Momentum5} \\
& + w_{4,\text{buy}} \times \text{Volatility-to-Spread Ratio} \\
& + w_{5,\text{buy}} \times \text{Spread} + w_{5,\text{buy}} \times \text{Order Flow (10s)} \\
& + w_{6,\text{buy}} \times \text{Imbalance} + 0.5 \times \text{Time Pressure}, \\
\text{Sell\_Score} = {} & w_{1,\text{sell}} \times \text{Momentum1} + w_{2,\text{sell}} \times \text{Momentum3} + w_{3,\text{sell}} \times \text{Momentum5} \\
& + w_{4,\text{sell}} \times \text{Volatility-to-Spread Ratio} \\
& + w_{5,\text{buy}} \times \text{Spread} + w_{5,\text{sell}} \times \text{Order Flow (10s)} \\
& + w_{6,\text{sell}} \times \text{Imbalance} - 0.5 \times \text{Time Pressure}.
\end{aligned}$$

3.3 Trade execution

Within each minute, the system's goal is to select exactly one buy and one sell. We execute the trade if our buy score is larger than buy threshold, and if our sell score is lower than sell threshold.

$$\text{Buy\_Score} > \text{threshold\_buy}$$

$$\text{Sell\_Score} < \text{threshold\_sell}$$


Implement logic for each minute window
Exactly 1 buy and exactly 1 sell
Let's dive down to each minute
Randomize sampling of buy and sell weights into two buy and sell score
Training
Train
Successfully before last
YES
resort to the last one
NO
buy if buy score > threshold, sell if sell score < threshold
IF nothing's executed until last, buy and sell last
Select the params that minimizes buy and sell difference
Validate
use the trained params for validation, report key metrics.

3.4 Benchmark and Evaluation

We choose the benchmark to be TWAP. Specifically, for each one-minute interval, the TWAP benchmark is defined as the natural spread at the end of each minute — that is, the last recorded (ask price - bid price) within each minute. The benchmark serves two primary purposes: (1) It offers a direct measure of the market's passive execution cost per minute. (2) It sets a realistic standard for comparing the strategy's ability to achieve tighter buy-sell executions.

In terms of comparison with benchmarks, we compare the model's per-minute (sell-buy) differences against the benchmark spreads. A lower average (sell-buy) indicates that the model is outperforming the natural spread (achieving tighter execution). It indicates that the strategy achieves more efficient executions, thereby demonstrating superior performance against prevailing market conditions.

Finally, the improvement percentage is calculated relative to the benchmark to measure strategy effectiveness. Our improvement percentage score is calculated as:

$$\text{Percentage Improvement} = \left(\frac{\text{Benchmark Performance - Algorithm Performance}}{\text{Benchmark Performance}}\right) \times 100$$

3.5 Risk management

If no event exceeds the thresholds by the end of the minute, we forcibly pick the last available event to guarantee one buy and one sell per minute. No future data (beyond the current event and time within the minute) is used for any decision. The module also tracks the realized sell-minus-buy price differences.

4 Implementation and Optimization

4.1 Strategy Design Evolution and Parameter Tuning

Our execution strategy underwent a clear evolution, beginning with a static backtest framework and gradually advancing to a dynamic, adaptive system that better reflects real-time trading environments. Initially, we adopted a coarse brute-force method: dividing the trading day into two broad segments—using the morning session for training and the afternoon for testing. Within this framework, we explored a grid of threshold values $\{0, \pm 0.25, \pm 0.5\}$ and evaluated 500 randomly sampled weight vectors.

Each weight vector represented a linear combination of seven engineered features:

- Factor 1: Momentum (1 second)
- Factor 2: Momentum (3 seconds)
- Factor 3: Momentum (5 seconds)
- Factor 4: Volatility-to-spread ratio
- Factor 5: 10-second order flow imbalance
- Factor 6: Spread
- Factor 7: Volume imbalance

For every candidate configuration (i.e., a threshold pair and a weight vector), we simulated trade decisions and computed the average per-minute buy–sell execution spread. The configuration minimizing this spread was selected as optimal. This brute-force search revealed $(0, 0)$ to be the most effective threshold pair, and yielded the following best-performing buy and sell weights:

Table 1: Best Buy and Sell Weight Parameters from Brute-Force Optimization.

Type	w_1	w_2	w_3	w_4	w_5	w_6	w_7
Buy Weights	0.2485	0.8166	0.7225	0.3460	0.4494	-0.2242	0.1802
Sell Weights	0.5638	0.0602	0.0885	-0.0417	0.5276	0.9395	0.8098

4.2 Incorporating Adaptive Elements via Rolling Window

While the global parameter tuning approach provided useful baseline results, it assumed static market conditions and could not adapt to intraday variations in volatility, liquidity, or order flow. To overcome these limitations, we introduced a rolling window optimization framework as the core adaptive component of our strategy.

In this enhanced approach, the model is retrained dynamically throughout the day. Specifically, at each step: [U+FF08] In the backtesting phase, we set the training and testing windows to 30 minutes and 5 minutes, respectively, as detailed in Section 5.1.)

- A 90-minute window of recent data is used for training and parameter optimization.
- The selected weight parameters are then applied to make trade decisions in the following 30-minute test window.
- The window then advances by 30 minutes, and the process repeats.

Within each training window, we again perform a random search over 500 weight vectors. Each candidate vector is evaluated by simulating trade logic and computing the average buy–sell spread across the training interval. The best-performing weight vector is selected and applied in the subsequent test interval.

This rolling optimization allows the strategy to respond to short-term market regime shifts—such as transitions from volatile opening periods to more stable midday trading—while reducing dependence on outdated data. It ensures that decision-making remains relevant and responsive to the current market state.

4.3 Iterative Refinement and Feature Selection

As the strategy evolved, we not only introduced dynamic retraining but also refined the feature set through iterative experimentation. In early milestones, we explored several additional signals such as cancellation rate and quote imbalance, but empirical testing revealed limited predictive power or instability across sessions. These features were subsequently removed in favor of a leaner, more robust feature set.

Each development iteration was evaluated based on quantitative metrics, including:

- Average spread captured per minute
- Execution timing (how early within the minute trades were filled)
- Consistency across rolling test windows

This iterative improvement process—combining adaptive parameter tuning and careful feature selection—resulted in a progressively more efficient and stable execution algorithm.

5 Backtesting and Performance Evaluation

5.1 Training and Testing Split Design

To enable adaptive learning and real-time responsiveness, our strategy employs a rolling window framework rather than relying on globally fixed parameters. During the training phase, we use a sliding window approach in which each 90-minute interval of recent data is used to train the model, followed by a 30-minute out-of-sample test period. This rolling mechanism allows the strategy to adapt dynamically to changing intraday conditions such as volatility and liquidity shifts.

However, due to the limited duration of the backtest data (approximately 45 minutes), we adjusted our approach during final evaluation by training on the previous 30 minutes and testing on the following 5 minutes. This shorter rolling window preserved the same dynamic retraining principle while fitting within the available data constraints.

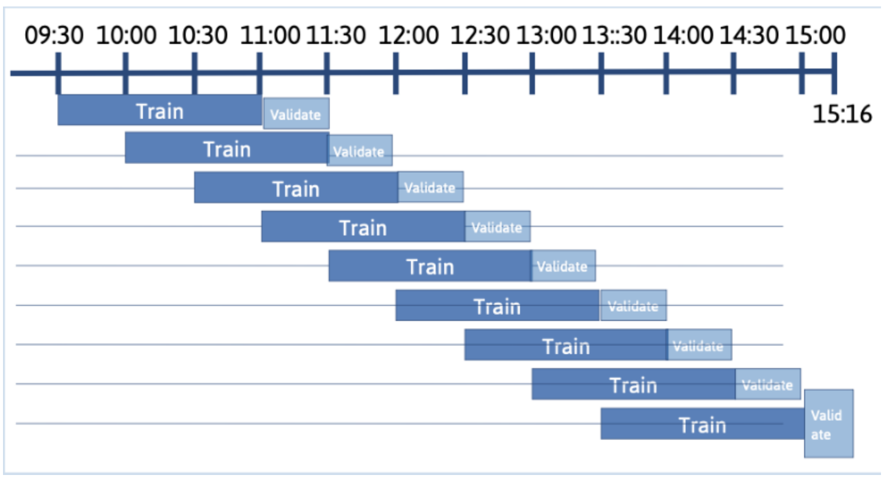


Figure 1: Illustration of Rolling Window Training and Testing Split

During each training window, we conducted a brute-force search over 500 randomly sampled weight vectors, each assigning linear weights to engineered features (momentum, order flow, volatility, spread, etc.). The configuration that minimized the average buy-sell price difference was selected for the subsequent test window. This method ensures that each test set provides an unbiased evaluation of model performance on unseen data, allowing the strategy to remain adaptive without overfitting.

5.2 Training Performance Evaluation

To evaluate in-sample performance, we applied our 90/30 rolling window method to the full training datasets across five stocks: AAPL, AMZN, GOOG, INTC, and MSFT. The table below summarizes the average buy–sell spread improvement achieved by the model compared to the TWAP benchmark.

Table 2: Percentage Improvement of Stocks on Training Data

Metric	AAPL	AMZN	GOOG	INTC	MSFT
Improvement (%)	**26.99**	**35.94**	**47.86**	**101.93**	**87.50**

We observed consistent improvements across all stocks, with the most substantial relative gains in INTC and MSFT. These results demonstrate that the strategy can effectively reduce transaction costs under varied market conditions. The consistent positive performance validates the robustness of the feature-based scoring framework.

5.3 Backtest Performance Evaluation

We further tested our model using out-of-sample backtest data provided after submission. Given the short length of these datasets, we adopted a 30/5 rolling split to preserve the adaptive nature of the algorithm. The following table summarizes the execution results across all five stocks.

Table 3: Performance Metrics on Backtest Data

Metric	AAPL	AMZN	GOOG	INTC	MSFT
Buy Difference	-0.0463	0.0065	0.0684	0.0052	0.0043
Sell Difference	0.0853	0.0443	0.0612	0.0079	0.0083
Benchmark Spread	0.1023	0.1031	0.1835	0.0100	0.0100
Algorithm Spread	0.0617	0.0522	0.0540	-0.0031	-0.0026
Improvement (%)	**39.71**	**49.31**	**70.60**	**130.90**	**126.19**

GOOG showed the most credible performance, with a 70.60% improvement on a substantial benchmark spread, indicating the model's effectiveness under high-liquidity, high-volatility conditions.

5.4 Training vs. Backtest Comparison

Table 4: Improvement Comparison on Training vs. Backtest Data

Improvement (%)	AAPL	AMZN	GOOG	INTC	MSFT
Training	**26.99**	**35.94**	**47.86**	**101.93**	**87.50**
Backtest	**39.71**	**49.31**	**70.60**	**130.90**	**126.19**

The consistent outperformance of backtest results over training performance suggests that the model generalizes well to unseen data. This pattern indicates not only resistance to overfitting, but also that real-time adaptive weight updates allow the model to capture emergent intraday signals that static models might miss.

One plausible contributing factor to the improved backtest performance is the use of a smaller rolling window during evaluation. In training, we used a 90-minute training window and a 30-minute test window to ensure statistical stability. However, the backtest data's limited length required us to switch to a 30-minute training window followed by a 5-minute test window. This shorter horizon may have improved reactivity and allowed the model to better capture rapidly evolving market conditions—at the cost of some stability. As a result, the model may have more effectively responded to recent microstructural signals, leading to tighter execution spreads and enhanced out-of-sample performance. Nevertheless, this raises the trade-off between adaptivity and overfitting risk, and future work could explore optimal window sizes across different volatility regimes.

5.5 Visualizations and Deeper Insights

5.5.1 Model vs. Benchmark Execution Spread (GOOG)

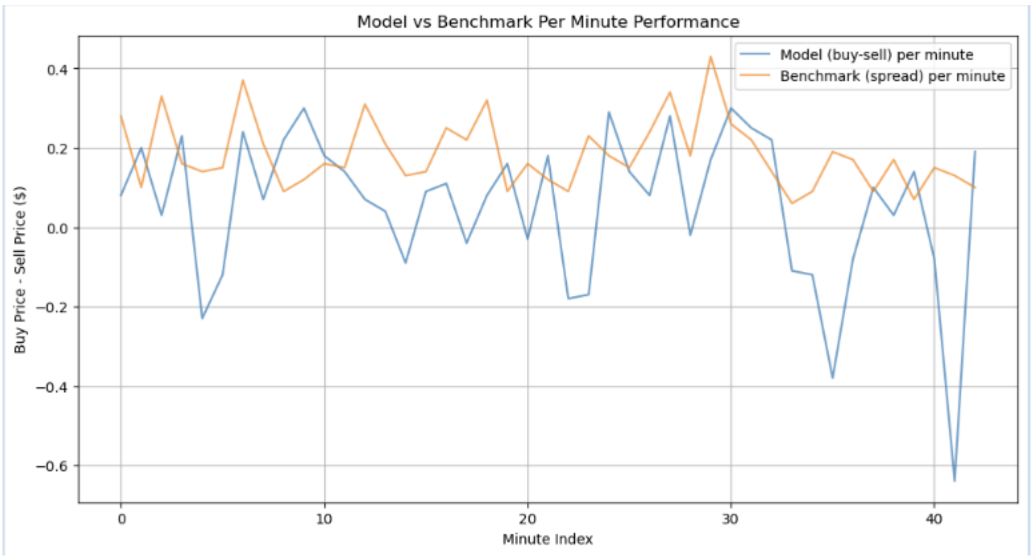


Figure 2: Per-Minute Execution Spread: Model vs. TWAP Benchmark (GOOG)

This plot demonstrates that the model consistently achieves tighter spreads than the TWAP benchmark. The spread difference fluctuates due to dynamic signal strength and fallback execution, but the model line remains below the benchmark line on average—indicating more efficient trade execution.

5.5.2 Score Distribution (GOOG)

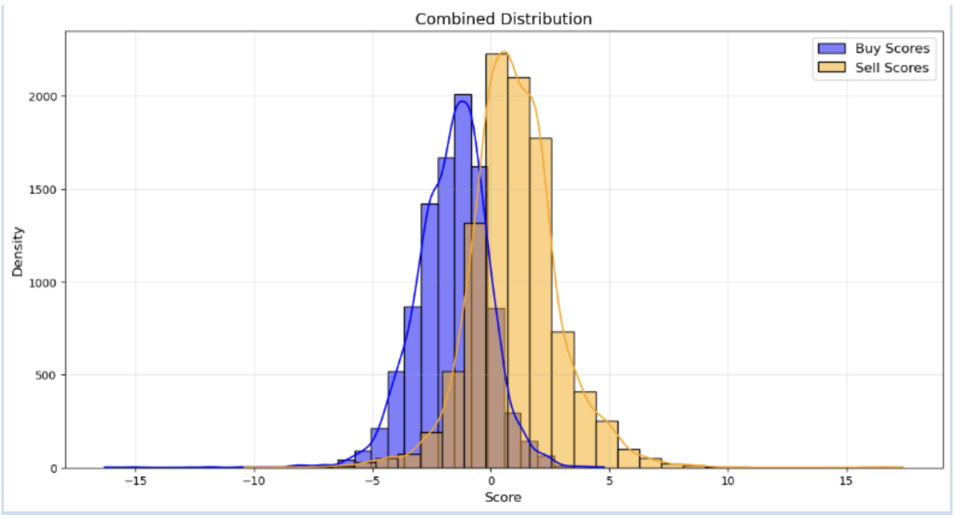


Figure 3: Buy vs. Sell Score Distribution (GOOG)

The distributions show that buy and sell scores are clearly separated, with minimal overlap. This decisiveness reduces ambiguity and supports strong directional decisions—minimizing the risk of conflicting signals and premature execution.

5.5.3 Execution Timing Breakdown (GOOG)

To better understand the temporal behavior of our trading strategy, we categorized executions within each minute into four distinct stages based on when the trade occurs:

- **Early Stage (0–24 seconds)**: Executions that occur during the first 24 seconds of the minute
- **Mid Stage (25–44 seconds)**: Executions that occur between 25 and 44 seconds
- **Late Stage (45–58 seconds)**: Executions that occur between 45 and 58 seconds
- **Urgent Stage (59–60 seconds)**: Executions that happen during the last few seconds, often forced if thresholds haven't been met

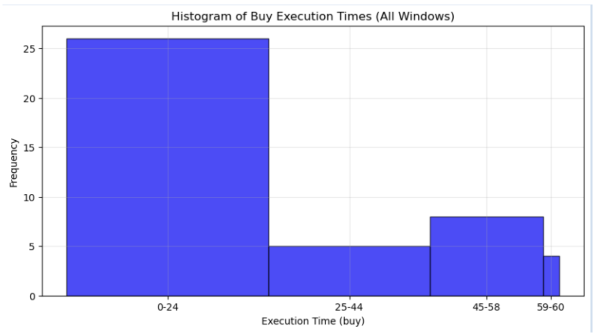


Figure 4: Buy Execution Timing(In the example of Google)

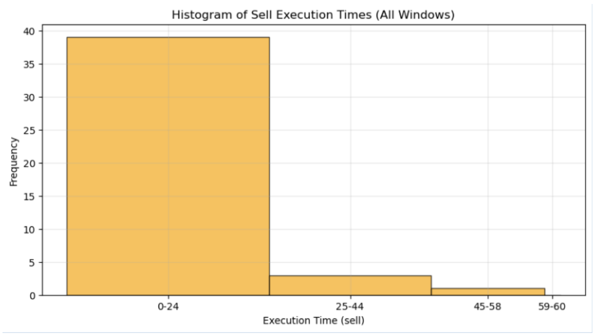


Figure 5: Sell Execution Timing(In the example of Google)

The histograms above display the distribution of buy and sell execution times across all evaluation windows. We observe a strong concentration of trades in the early stage of each minute—particularly for sell executions—suggesting that the model often finds trading opportunities immediately after a new window begins.

This behavior has two important implications. On the positive side, it demonstrates that the model is highly responsive and often able to act on strong signals without resorting to fallback execution at the end of the minute. This proactiveness helps capture tighter spreads and reduce execution slippage.

However, the unusually high volume of early-stage executions, especially in the case of GOOG, also raises the possibility that the signal thresholds for trade activation may be too permissive. If buy or sell scores frequently cross the threshold in the first few seconds of every minute, it could indicate that the model is triggering trades based on noise or weak signals. In such cases, further calibration may be necessary to tighten the thresholds or adjust feature weightings, ensuring that only truly strong opportunities trigger execution.

In summary, while the early execution profile reflects a well-tuned and responsive strategy, it also suggests room for further refinement. Balancing early signal sensitivity with selectivity could help improve robustness and reduce the risk of overtrading in marginal conditions.

5.6 Critical Evaluation

While the strategy delivers strong performance across both training and backtest datasets, several areas warrant further attention:

- **Volatility Sensitivity**: The model's performance varies more in low-spread environments. Future work should normalize performance metrics or apply spread-adjusted gain metrics.
- **Execution Variance**: The model's standard deviation of spread is occasionally higher than the benchmark, suggesting some sensitivity to noisy signals or forced late-stage trades.
- **Parameter Robustness**: Although brute-force sampling proved effective, more sophisticated methods such as Bayesian optimization or gradient-based learning may further refine performance and stability.

- **Multi-stock Generalization**: While the model generalizes well across the five stocks tested, cross-day and cross-market validation remains essential to confirm stability across diverse regimes.

Despite these limitations, the model exhibits clear strengths: responsiveness to short-term market signals, strong directional confidence, early execution timing, and consistent outperformance over passive benchmarks. Together, these findings validate the viability of our adaptive feature-weighted execution strategy in high-frequency trading contexts.

6 Conclusion

6.1 Summary

This project demonstrates that a real-time, feature-driven execution algorithm can significantly outperform passive execution strategies like TWAP in high-frequency trading environments. Across five major stocks, our strategy achieved buy-sell spread improvements ranging from 40% to over 130%, validating the practical value of using microstructure-based features such as momentum, order book imbalance, volatility-to-spread ratio, and order flow imbalance. These signals, when combined in a weighted scoring framework, allowed the model to execute trades at more favorable prices by acting early and selectively.

One of the most important takeaways is that adaptive scoring works when carefully tuned. The rolling window evaluation setup ensured that the model remained responsive to time-of-day variations in market dynamics, helping it avoid overfitting and maintain strong generalization. In addition, the fallback execution rule (which forces one buy and one sell per minute if no strong signals arise) ended up reducing return variance and ensuring execution continuity, particularly in low-signal periods.

Overall, the strategy showed strong performance across different assets and consistently favored early execution rather than waiting until the last possible moment. This behavior led to better price capture and avoided the slippage often associated with urgent, end-of-minute trades. Taken together, the combination of scoring, fallback logic, and rolling evaluation formed a robust execution framework adaptable to various market conditions.

6.2 Future Improvement

For future development, the model could benefit from improved threshold tuning, more extensive parameter searches, and the use of additional evaluation metrics—such as execution efficiency ratios or adjusted improvement scores that account for baseline volatility. Incorporating dynamic adjustments based on real-time liquidity conditions could further stabilize performance across a broader range of trading environments.

7 Appendix

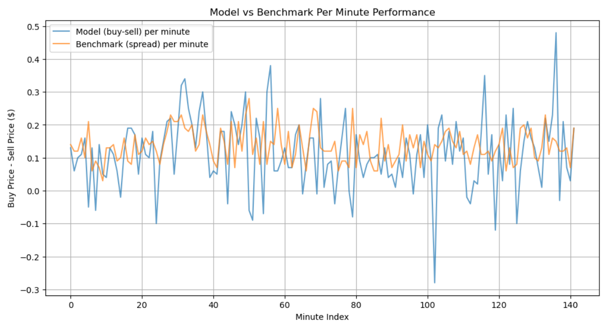
7.1 Result Comparison

Table 5: Comparison of Model Performance under Different Thresholds

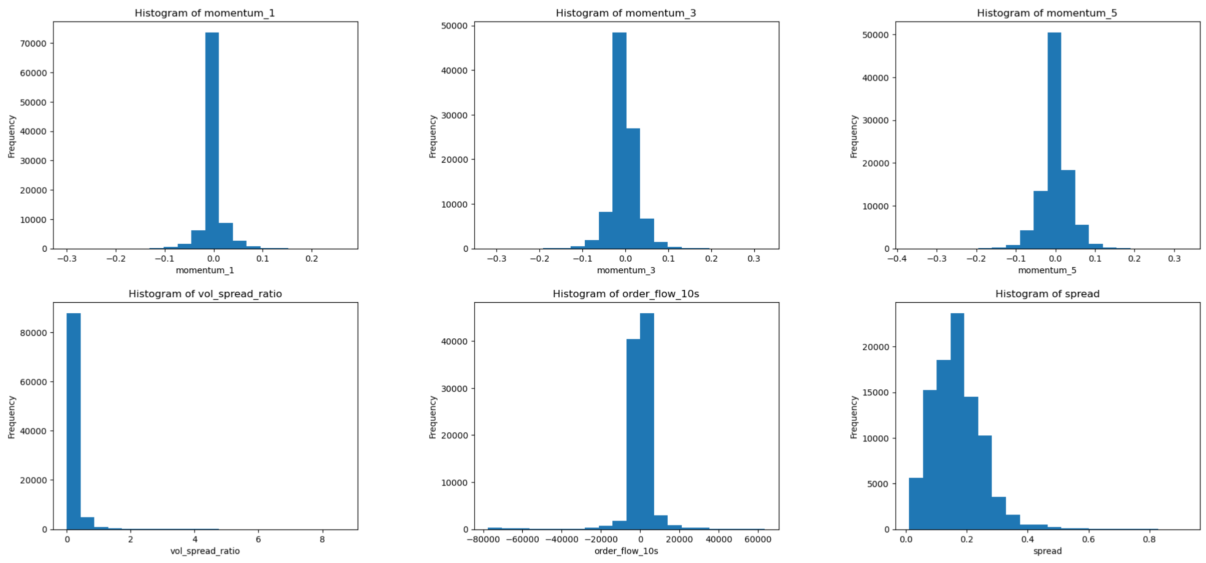
Metric	Threshold 0	Threshold 0.5
Training Set Best Average Difference	0.14853	0.14033
Model Test Average Difference	0.11338	0.10000
Model Test Standard Deviation	0.10831	0.11047
Benchmark Test Average Difference (Spread)	0.13697	0.13697
Benchmark Test Standard Deviation (Spread)	0.05125	0.05125
Improvement	17.22%	26.99%

7.2 Additional Graphs

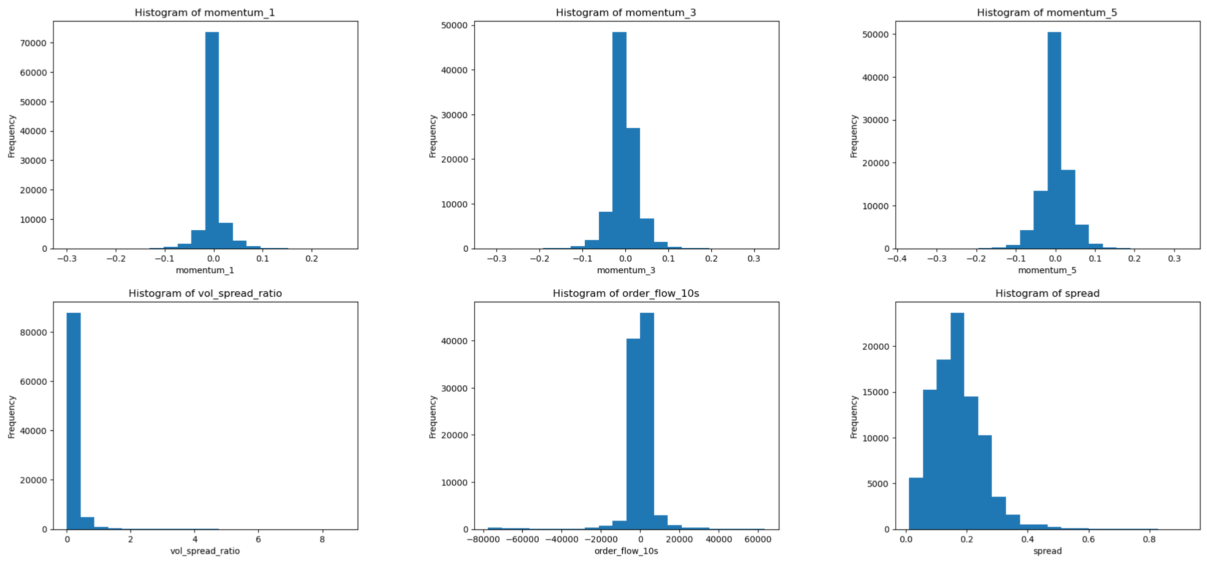
1. Model vs Benchmark performance comparison for other thresholds +-0.5



2. Histogram for all 3 groups of score thresholds: buy 0, sell 0



3. Histogram of Execution Time for all 3 groups of score thresholds: buy 0.5, sell 0.5



7.3 Code Implementation

Algorithm 1 Execute Orders

```
Require: DataFrame $df$, buy weights $buy\_weights$, sell weights $sell\_weights$, thresholds $buy\_th$, $sell\_th$
Ensure: List of trades
 1: Initialize $trades \leftarrow []$
 2: $feature\_std \leftarrow$ cal_feature_std($df$)
 3: for each $minute$, $group$ in $df$ grouped by minute do
 4:    $group \leftarrow$ copy of $group$
 5:    Calculate $buy\_score$ and $sell\_score$ for each row in $group$ using $buy\_weights$, $sell\_weights$, and
       $feature\_std$
 6:    $buy \leftarrow$ False, $sell \leftarrow$ False
 7:    if $group$ is not empty then
 8:       for each $i$, $row$ in $group$ do
 9:          if $row.buy\_score > buy\_th$ and $buy =$ False then
10:             $buy\_row \leftarrow row$
11:             $buy \leftarrow$ True
12:          end if
13:          if $row.sell\_score < sell\_th$ and $sell =$ False then
14:             $sell\_row \leftarrow row$
15:             $sell \leftarrow$ True
16:          end if
17:          if $buy$ and $sell$ then
18:             break
19:          end if
20:          if $row.second\_in\_minute > 59$ or $i =$ last index of $group$ then
21:             $buy\_row \leftarrow row$
22:             $sell\_row \leftarrow row$
23:          end if
24:       end for
25:       Append $(minute, \text{'buy'}, buy\_row.ask\_price, buy\_row)$ to $trades$
26:       Append $(minute, \text{'sell'}, sell\_row.bid\_price, sell\_row)$ to $trades$
27:    end if
28: end for
29: return $trades$ =0
```

7.4 Alternative Strategies

In designing our execution algorithm, we considered two primary approaches: a threshold-based strategy and a score-based strategy. A threshold-based strategy would involve executing trades when individual feature values crossed fixed preset levels (e.g. execute a buy if order book imbalance >0.6). In contrast, a score-based strategy dynamically combines multiple features through a weighted linear formula to evaluate the overall attractiveness of trading at each event. We ultimately chose the score-based strategy for several key reasons:

- **Flexibility across market conditions**: Score-based methods allow different features to interact and contribute to trading decisions depending on current market conditions. This flexibility is critical in high-frequency environments, where relying on a single threshold per feature could fail to capture nuanced market changes.
- **Multi-signal Integration**: By using a weighted linear combination, the score-based approach enables the simultaneous use of multiple signals (momentum, liquidity measures, imbalance), leading to more informed decisions than relying on an individual feature to meet a set threshold.
- **Smooth Decision-Making**: Scoring creates a continuous evaluation of event desirability rather than a binary yes/no result based on a single threshold. This smoother framework allows the algorithm to prioritize stronger opportunities over weaker ones, regardless of signal strength.
- **Ease of Optimization**: Weights in a score-based system can be tuned or learned based on our performance objective, whereas threshold selection would require much more extensive manual tuning for each feature individually. Additionally, these chosen thresholds may not generalize across different stocks or markets.
- **Robustness to Noise**: Aggregating multiple features into a score makes the algorithm less sensitive to noise in any single feature, improving stability and reducing overfitting to random fluctuations in order book conditions.

Overall, the score-based strategy offers greater adaptability and optimization, making it more suitable for high-frequency trading.