University College London

Department of Computer Science

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computational Finance, University College London

Predicting Return Direction from Volume-Sampled Limit Order Books: A Feature Engineering Approach

Author
Artem Arshakyan

Academic Supervisor
Dr Silvia Bartolucci
Department of Computer Science
University College London

Industrial Supervisor
Mr Eric John
HSBC

September 29, 2025

This dissertation is submitted as part requirement for the MSc Computational Finance degree at UCL. It is substantially the result of my own work except where explicitly indicated in the text.

Abstract

High-frequency trading (HFT) lacks a single, concrete definition. Most accounts emphasise its quick execution speeds and, some, a shift from clock time to a "volume clock." Although HFT operations are highly secretive, the very fact that such firms profit at these horizons stands in tension with the strong form of market efficiency. Such firms find reason to depart from the market portfolio, consequently generating small, consistent *alpha*. Motivated by this insight, we assess the scope for alpha generation in a high-capitalisation stock. We engineer domain-informed features from LOB snapshots and event streams in volume time, casting the task as a three-class classification problem (down/flat/up). Our results indicate weak but genuine class structure, with models consistently more reliable on one minority class. The existence of label/feature structure is corroborated by permutation tests, and model attributions consistently assign the greatest influence to bar (executed) return—consistent with well-documented short-horizon negative autocorrelation in returns. Importantly, removing this feature still yields statistically significant structure, implying that our models leverage other bespoke features in concert with bar return: the microstructure regularity is not the sole driver. Overall, the signal tends to lower cross entropy while hurting accuracy, suggesting that bar return often nudges predictions in the right direction, whereas secondary features add only modest—often noisy—incremental information, consistent with the well-known difficulty of alpha generation in the absence of under-exploited data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

High-frequency trading (HFT) typically refers to sophisticated, fully automated trading systems characterised by quick execution speeds and short holding horizons. These systems operate in a quantitatively driven paradigm and often learn from time series observed at *nanosecond* scales where the signal-to-noise ratio is low—much of the noise arising from *market microstructure*. The structure of this noise depends critically on the exchange's trading protocol; most modern venues run a continuous double auction implemented via a limit order book (LOB).

Consequently, a solid grasp of LOB dynamics is essential for training useful models downstream in many strategy pipelines: it sharpens the modeller's intuition about the bespoke inductive biases the architecture should encode. While model choice matters, the dataset and problem formulation matter more. Careful, domain-informed data curation is what instantiates these inductive biases and materialises *information arbitrage* (Chollet, 2021, Ch. 10, p. 308), enabling HFT firms to generate *alpha*—mathematically, positive risk premia relative to a market portfolio (commonly proxied by an index such as the S&P 500).

The pursuit of alpha, however, sits in tension with the efficient market hypothesis (Fama, 1970), especially in its strong form, which posits that all public *and* private information is fully reflected in prices—implying the market portfolio cannot be systematically beaten. In practice, many investors therefore adopt passive vehicles (e.g., ETFs); indeed, "passive investing has risen from about 1% of the total in the early 1990s to over 50% by 2024.[1]" Yet the continued presence of informed active investors suggests that enough opportunities exist where the benefits of beating the market outweigh the costs—Pedersen (2019) coins this "efficiently inefficient." Paradoxically, it seems it is such informed traders that help enforce market efficiency by injecting their information into prices and shaping price formation.

Alternative formulations of HFT emphasise a *sampling paradigm* rather than speed.

[1] https://www.lse.ac.uk/finance/news/2024/Research-Showcase-PassiveInvesting

Easley et al. (2012) define HFT through a shift to a volume-based clock. While acknowledging that speed aids in generating "tiny predictive power on a sufficiently large number of independent bets," they argue HFT would persist even without the extreme latency advantages, simply due to the event-based sampling shift. Adopting a volume-based sampling mechanism helps avoid over- and under-sampling across inactive and active periods and can yield return increments that are closer to Gaussian, as stated by Mandelbrot and Taylor (1967). We take inspiration from this approach—further refined in López de Prado (2018)—and investigate alpha generation in equity LOBs using heavy feature engineering and volume-based sampling.

Specifically, we conduct a case study on Microsoft (MSFT)—the second-largest company by market capitalisation—to examine whether *alpha* can be extracted from raw equity LOB data. Our data come from LOBSTER, which reconstructs the Nasdaq LOB and provides both event streams and depth snapshots. We convert tick-time data to a *volume-clock* (volume-bar) representation, where each bar aggregates trades until 10,000 MSFT shares have been executed. Features are then constructed after the accumulation of these trades from the LOBSTER event streams and LOB snapshots. We restrict attention to depth-5 books: as Tran et al. (2022) note, informativeness declines rapidly beyond the top levels, with the first four carrying most of the signal; we include level 5 as a one-level "grace" buffer.

Evidently, beating the market is hard. López de Prado (2018) emphasises that success typically stems from access to difficult, under-exploited data—an idea aligned with Chollet (2021)'s notion of *information arbitrage*. LOBSTER, however, is widely used. Thus, **if** a statistical edge exists here, it would likely arise from *problem formulation*: careful feature construction, sampling choices, and label generation. Accordingly, our methodology prioritises the design of the learning task and data curation. We adopt a fixed-horizon, three-way labelling scheme based on the close–close executed return of each volume bar. The label is determined by whether this return exceeds, is below, or lies within an exponentially weighted moving standard deviation (EWMSD) threshold. The resulting task is therefore a three-class classification problem.

To evaluate label–feature informativeness, we use the *Rough Transformer* of Moreno-Pino et al. (2025), which augments the standard Transformer by replacing the embedding matrix with rough path signatures that encode long- and short-term feature interactions. We adopt this architecture for its ability to capture long-range dependencies—assumed to be prevalent in LOB dynamics—and to inject feature interactions directly via signatures (see Section 3.3.1). Because signature computation is defined on continuous paths via iterated integrals, discrete samples must first be lifted to a continuous representation. Following Moreno-Pino et al. (2025), we use piecewise–linear interpolation; we also uniquely extend this with a smoothing spline interpolation. In addition, we benchmark the Rough Transformer against alternative temporal models and a common model in financial

machine learning—the Random Forest (López de Prado, 2018)—all evaluated *without* a signature transform.

To assess whether the engineered features and labels possess genuine class structure, we employ a *cyclic permutation test* on the labels, testing the null of feature–label independence by cyclically rotating labels relative to the feature sequences. Conditional on rejecting the null, we then analyse SHAP values for the best model on the test set. As in many financial machine learning settings, interpretability can be more informative than model performance—apparent performance may reflect sampling variability, which we try to mitigate with a very large time span (5 months). We find that a single feature—Execution Return—appears most informative and dominates the others. However, when we omit it from our best model (the LSTM), the permutation-test null is still rejected under an FDR-controlled multiple-testing framework, indicating that other—albeit much weaker—signals are present.

Concretely, we address the following research questions, which build sequentially on one another:

- **RQ1:** Can we make consistently accurate directional predictions on MSFT LOBSTER data using volume-based sampling and engineered features?
- **RQ2:** Is there genuine class structure in our data, or can comparable results be obtained under random feature/label alignment?
- **RQ3:** Which aspects of the engineered features does the model exploit to guide its predictions?

To the best of our knowledge, this is the first study to apply the Rough Transformer to LOB data, despite LOBs frequently being cited as a motivating problem domain. Moreover, we introduce a novel extension in which path signatures are computed on a smoothing–spline approximation rather than via piecewise–linear interpolation; we refer to this variant as the *Spline Transformer* throughout.

Chapter 2

Literature Review

2.1 Neural Networks & The Rough Transformer

The universal approximation property of neural networks dates to the late 1980s. In a seminal result, Cybenko (1989) showed that a single–hidden–layer feedforward network with a sigmoidal activation can approximate *any continuous function* on the unit hypercube to arbitrary precision. Subsequent work by Hornik (1991) established that this universality does not depend on the specific sigmoid choice: for *bounded, continuous, non-constant* activation functions and inputs lying in a compact subset of $\mathbb{R}^n$ (closed and bounded), multilayer feedforward networks are universal approximators—that is, they are dense in $C(X)$ for compact X. Similar results are proven for other classes of activation functions (notably, like $\mathrm{ReLU}(\cdot)$).

Universality, however, is a double-edged sword. A weak inductive bias implies a vast hypothesis space—powerful but easy to overfit—so optimisation and data curation become critical. As Chollet (2021, Ch. 10, p. 290) notes, deep learning representation search can resemble "searching for a needle in a haystack," and hence can still fail to beat simpler models. Relatedly, they mention the *latent manifold hypothesis* (Chollet, 2021, Ch. 5, p. 128), which posits that natural data typically concentrate near a low-dimensional manifold within the ambient space.[1] Effective learning, therefore, hinges on informative features that expose this manifold structure (our natural data). In practice, careful data preparation and feature engineering are indispensable, let alone in a low signal-to-noise ratio domain such as the financial markets; therefore, López de Prado (2018), for the majority of his book, discusses data curation and financial market feature engineering. Our data curation vastly draws on these principles—including volume sampling, feature engineering, and label construction—as well as domain insights from the canonical LOB introduction of Gould et al. (2013).

[1] Think of a univariate OLS with high R^2: the data live in $\mathbb{R}^2$, yet a one-dimensional subspace explains most of the variation.

As mentioned, we consider neural networks mainly within a Transformer architecture. An encoder-only Transformer is most familiar from NLP. A sentence is split into sub-word tokens, each token is mapped to a learned embedding vector, and a learned *positional embedding* is added so the model knows where each token sits in the sequence (segment embeddings exist but are ignored here). Each *encoder block* contains two sub-layers: (i) multi-head *self-attention* and (ii) a position-wise *MLP* (two linear layers with a nonlinearity such as $\mathrm{ReLU}(\cdot)$), each wrapped with a *residual connection* and *layer normalisation*; dropout is typically applied to attention weights and MLP activations. Vaswani et al. (2023) provides the canonical introduction.

We follow the work of Moreno-Pino et al. (2025), and extend the Transformer architecture into a time series domain. In their paper, they introduce the Rough Transformer. As the name suggests, this architecture heavily hinges on rough path theory for its formulation, and a cornerstone to rough path theory is the concept of a path's *signature*.

The path signature, introduced by Chen (1958) and later developed within rough path theory (see, e.g., Lyons (2014)), compresses the realised information of a path into a sequence of iterated integrals. For sufficiently regular paths (e.g., bounded variation), the signature determines the path up to measure zero sets. This signature calculation can be taken to arbitrary precision, and, by analogy with a Taylor expansion, truncating the signature at level n yields a finite-dimensional representation whose accuracy improves as n increases.

Moreno-Pino et al. (2025) replace the standard Transformer embedding matrix with representations built from *rough path signatures*. They term the resulting paradigm *multi-view signature attention*, which concatenates *global* and *local* path information: the global component aggregates long-range structure, while the local component emphasises recent dynamics. To deploy signatures on discretely observed time series, they linearly interpolate between samples—primarily to reduce computational burden—while noting that alternative interpolation schemes (e.g., spline-based) are plausible.

A key advantage of their method is its mitigation of the $\mathcal{O}(L^2)$ attention complexity, where L is the sequence length. Rather than attending over all observations, they subsample by choosing a fixed number of points—hyperparameterised as the *number of windows*—and compute global and local signatures on this reduced trajectory. This yields an effective length $\bar{L}$, substantially lowering cost while preserving essential dynamics.

Summarising, they approximate the feature-generating path via linear interpolation, use hyperparameter tuning to select which sample points to include, and then capture long- and short-horizon information across segments. Figure 3.3 illustrates the procedure on eight samples from a $\sin(t/2)$ data-generating process. The interpolated paths highlight segments whose geometric information is subsequently compressed within the multi-view signature attention framework.

Moreno-Pino et al. (2025) state that the Rough Transformer is well suited to long–range

dependent, irregularly sampled time series—such as LOB data—while remaining *sample-efficient*: it accelerates training yet retains much of the accuracy attainable on the full path. They attribute their architecture's success—achieving SOTA results while reducing computational burden—to *joint temporal and spatial compression*. Temporally, long-horizon behaviour is summarised with minimal information loss; spatially, cross-feature terms in the signature directly account for feature relationships. As detailed in our technical formulation (see Section 3.3.1), level $n > 1$ signatures involve iterated path integrals over n combined features, naturally capturing interaction-type effects. The authors offer intuition via a fully connected feature graph, where higher-order terms encode multiway feature interactions.

To evaluate these claims on LOB data, we compare the Rough Transformer against several temporal baselines and a Random Forest—the latter motivated by the bagging ensemble's ability to mitigate overfitting López de Prado (2018, Ch. 6, p. 98). Moreover, we consider a smoothing spline formulation (Reinsch, 1967), which fits a curve to the data while controlling roughness via a smoothing parameter (larger values yield trajectories that are less reactive to individual observations).

2.2 Permutation Tests

Ojala and Garriga (2009) provide a framework to assess whether multiple classifiers exploit statistically significant structure between labels and features using a permutation test that is model-agnostic and robust to class imbalance. Their work formulates two nonparametric hypothesis tests: (i) a permutation test of the null that labels are independent of features, and (ii) a stronger, within-class permutation that tests the null of mutual independence among features *conditional* on class.

In practice, Ojala and Garriga (2009) estimate p-values by first training the classifier on the original data to obtain a reference loss, then repeatedly retraining on datasets with permuted labels to form a null distribution of losses; the reference loss is compared against this null to compute the p-value. Because classifier losses can be seed-sensitive (e.g., from random initialisations/updates or feature subsampling in Random Forests), they recommend performing multiple "true" fits with different seeds, computing a p-value for each, and averaging these p-values for valid inference.

When assessing many hypotheses simultaneously, they control the false discovery rate (FDR) using the Benjamini–Hochberg procedure (Benjamini and Hochberg, 1995) at level $\alpha = 0.05$. This approach limits the expected proportion of false positives while avoiding the excessive conservatism of dividing α by the number of tests. Notably, the label-permutation test attains high power at large sample sizes, which we believe can render FDR control overly conservative; nevertheless, we apply the same procedure for consistency.

Finally, the two tests induce different permutation spaces, and in practice, the first

test tends to yield much smaller p-values than the second. Consequently, they claim it is sensible to proceed to the stronger within-class test only if the classifier first rejects the independence null in the basic permutation test. To note, however, a standard permutation test assumes an exchangeability hypothesis Valente et al. (2021), which is often violated in time series data. Kingsbury et al. (2019) circumvent this however by using a cyclical permutation.

2.3 SHAP Values

Once class structure is established in the dataset, Lundberg and Lee (2017) introduce SHAP values as a way to interpret those complex model predictions. They define an explanation as an additive feature-attribution model and show that the class of such methods satisfying *local accuracy* (i.e., $f(x) = g(z') = \phi_0 + \sum_{i=1}^{n} \phi_i z'$, formalised in Section 3.6.1), *missingness* (absent features receive zero attribution), and *consistency* (if a model change increases a feature's marginal contribution, its attribution does not decrease) has a unique solution: the Shapley values of the model's conditional expectation, linking to the cooperative-game formulation of Shapley (1953). Because exact computation requires enumerating all feature combinations, $\mathcal{O}(2^n)$ for n features, approximations are used in practice. One such method is *GradientSHAP*, which approximates SHAP by averaging *Integrated Gradients* (IG) along straight-line paths from baselines x' (sampled from a training-derived background) to the input x (Sundararajan et al., 2017). IG defines attributions via a path integral from the baseline (e.g., a "blank" image in CNNs) and satisfies key axioms such as *Sensitivity*, *Implementation Invariance*, and *Completeness*.

2.4 High–Low Variance Estimators

Before proceeding, we note that our feature engineering includes two relatively unorthodox features: the aggressor-side entropy rate and a high–low variance estimator. The entropy-rate construction is technical, so we defer it to Section 3.2.3.1. Below, we summarise relevant research on the high–low variance estimator.

There are several high–low volatility estimators (mentioned in Beckers (1983)) motivated by the idea that the price *range* within a period captures intra-period variability more effectively than a close–to–close change. The most widely known is Parkinson's estimator (Parkinson, 1980).

In their paper, Beckers (1983) compare Parkinson's estimator,

$$\sigma^2_{\text{Parkinson}} = \frac{1}{4n \ln 2} \sum_{t=1}^{n} \left(\ln \frac{H_t}{L_t} \right)^2, \tag{2.1}$$

where H_t and L_t are the bar's high and low, with the standard close–close volatility,

$$\sigma^2_{\text{CC}} = \frac{1}{n-1}\sum_{t=1}^{n}(C_t - \bar{C})^2,$$

where $\bar{C}$ is the mean close over the period. Now, under the assumption of a continuous price path, Parkinson showed that the relative efficiency of the high–low estimator versus the unbiased close–close estimator is

$$\text{Eff} = \frac{\text{Var}(\widehat{\sigma}^2_{\text{CC}})}{\text{Var}(\widehat{\sigma}^2_{\text{Parkinson}})} = 5.2.$$

Beckers (1983) investigate whether, when these assumptions are violated, the Parkinson estimator contains information not captured by the close–close measure. They run two single-variable regressions of each estimator on its own lag, and two more on the other estimator's lag. The Parkinson estimator exhibits higher R^2 throughout (which, in a univariate OLS with intercept, equals the squared Pearson correlation), indicating lower variance than the close–close estimator and suggesting uncaptured information in the range. This is all, however, at the cost of slight bias.

The use of such an estimator is further supported by Gould et al. (2013), who note that $\ln(H/L)$ provides a simple volatility proxy—which is proportional to the square root of Parkinson's estimator when $n = 1$.

Chapter 3

Methodology

3.1 Background on Limit Order Books

Most exchanges (e.g., NYSE, Nasdaq, LSE) implement a continuous double auction via a limit order book (LOB). The LOB accumulates bids and asks on either side, reflecting the net declared trading desires at a given time. Specifically, with a slight abuse of notation, the state of an LOB at time t is

$$\mathcal{L}(t) = \Big[P_i^b(t), V_i^b(t), P_i^a(t), V_i^a(t)\Big]_{i=1}^{K},$$

characterised by the top K price levels (depth), $P_{\leq K}^{\text{side}}$, and their available sizes, $V_{\leq K}^{\text{side}}$, on the bid (b) and ask (a) sides.

At each price level, the book aggregates outstanding limit orders. We represent a limit order by $l = (t, q, p)$, meaning that at time t the trader wishes to buy or sell q units at a price no worse than p. Prices and sizes are quantised by the market's *resolution parameters*: the tick size π and the lot size σ. Accordingly, admissible quotes and sizes satisfy $p = \tilde{k}\,\pi$ with $\tilde{k} \in \mathbb{N}$ and $q = k\,\sigma$ with $k \in \mathbb{Z}$. In contrast to Gould et al. (2013), we adopt the LOBSTER convention: positive sizes denote buy limit orders and negative sizes denote sell limit orders. We use this convention in our feature definitions.

A buy limit order with $p \geq P_1^a(t)$ is termed a *market order* and executes immediately against the ask side up to

$$\min\Big(q, \sum_{\{j:\ P_j^a(t) \leq p\}} V_j^a(t)\Big).$$

Symmetrically, a sell market order with $p \leq P_1^b(t)$ executes against the bid side up to

$$\min\Big(|q|, \sum_{\{j:\ P_j^b(t) \geq p\}} V_j^b(t)\Big).$$

Orders with $p < P_1^a(t)$ (buy) or $p > P_1^b(t)$ (sell) rest in the book at their quoted level until

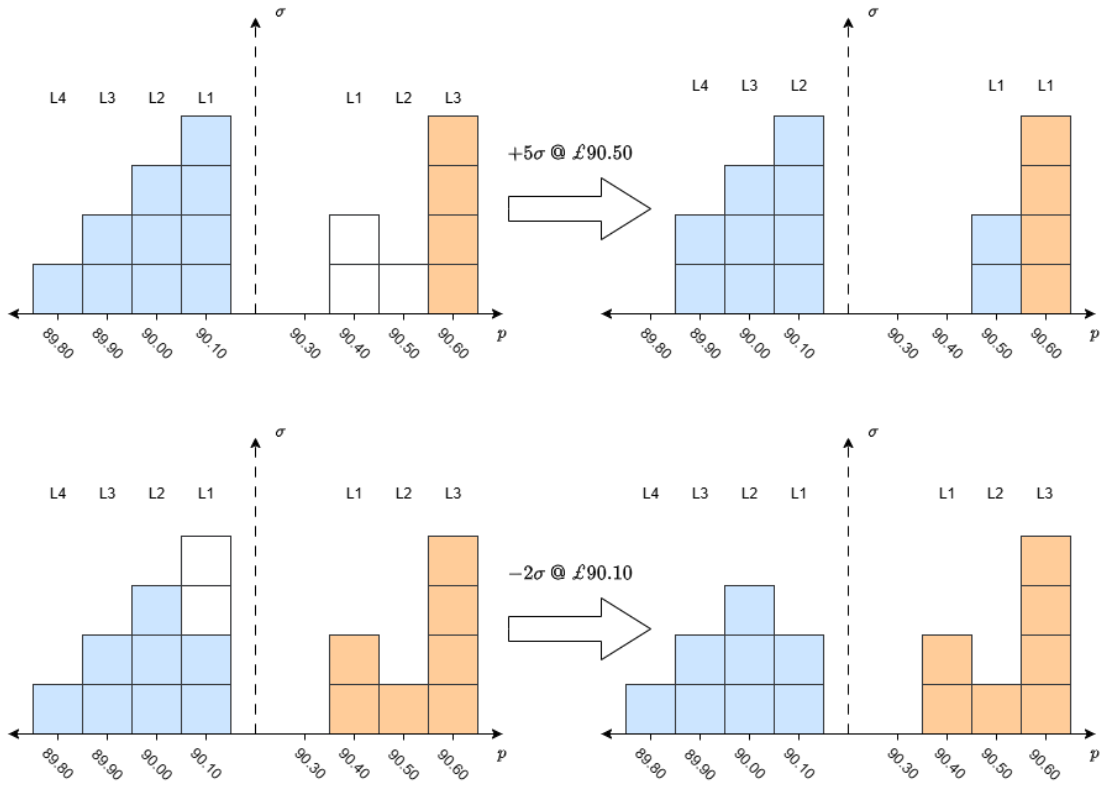


Figure 3.1: Effect of two incoming orders on the limit order book $\mathcal{L}(t)$. Blue denotes bid limit orders, orange denotes ask limit orders, and white marks resting orders that will subsequently be executed. The tick size is $\pi = £0.10$. On each side, only the top four price levels within the displayed range are shown, labelled L1–L4. The top panel corresponds to $l = (t, 5\sigma, £90.50)$; the bottom to $l = (t, -2\sigma, £90.10)$.

matched, modified, or cancelled. A concrete example is given in Figure 3.1.

The most important depth is Level 1, proxying liquidity via the bid–ask spread $\big(P_1^a(t) - P_1^b(t)\big)$ and informing price discovery via the mid-price $\big(P_1^a(t) + P_1^b(t)\big)/2$. Each subsequent level presents a correspondingly less favourable trade and, as discussed, wanes in importance with depth.

Within a price level, most venues use price–time priority: better prices rank ahead of worse; within a price, earlier timestamps fill first. Where hidden/iceberg orders are allowed, these typically receive time priority only for their displayed portion.

Many exchanges run short call auctions at the open and close (e.g., Nasdaq Opening/-Closing Cross). During the call, participants submit and cancel orders, which accumulate throughout the period. The uncrossing price $\widehat{p}$ is chosen to maximise matched volume. These auctions are Walrasian in spirit (single clearing price); the regular session is a continuous price–time-priority market.

Because the opening and closing auctions cover only a brief fraction of the day and have different mechanics from the continuous LOB, this thesis excludes auction intervals and analyses continuous trading (regular hours) only, ignoring trading halts.

The interested reader is directed to Gould et al. (2013) for a more in-depth overview.

3.2 Data & Feature Engineering

3.2.1 Raw Data & Volume Bar Construction

We collect Level-5 LOBSTER data for the second-largest stock by market capitalisation (cap) as of July 2025[1], MSFT, as stated here. We chose the second-largest market cap stock for this case study because the current largest, NVDA, was not available for download from LOBSTER for our desired period, spanning 2 January 2025 to 30 May 2025. LOBSTER provides two datasets, which we refer to as the "raw datasets" hereafter: a *message file*, recording event-level information (timestamp, event type, order size, and price), and the corresponding *order book file*, which provides the state of the book immediately after each event. The message file is central to our feature-engineering method; accordingly, we concatenate the two files and align the event stream with the corresponding state-effect stream.

Next, to construct volume bars, we estimate the average daily trading volume over this period by summing the monthly totals shown here and dividing by the number of trading days over the same interval, as listed by the New York Stock Exchange (NYSE). Although inter-monthly variation in volume can be high, this serves as a pragmatic reference point for our dataset generation, as we define volume buckets that accumulate traded shares in proportion[2] to this value. We balance two objectives: (i) larger buckets yield more reliable statistics, as many of our estimators converge asymptotically with trade count (e.g., aggressor entropy rate, Table 3.1); and (ii) smaller buckets increase the number of observations available to train our data-intensive neural networks. For MSFT, whose average daily volume is approximately 24 million shares, a bucket size of 10,000 shares provides a reasonable trade-off.

Our computationally intensive procedure is summarised in Algorithm 1. We iterate through the concatenated raw datasets, collecting the stream until the sum of shares traded for both hidden and displayed orders exceeds the set threshold. We then compute the features summarised in Table 3.1 on the resulting data slice, after which we discard the current volume slice and collect the next. This procedure first produces 35,655 volume bars and 16 features. After applying moving statistics for both features and labels (which require a look-back window), the final dataset contains 35,555 volume bars with a revised set of 16 features.

[1] In this section, any statistic that depends on the current time refers to July 2025.
[2] Anyway, not all traded volume is routed through Nasdaq.

Algorithm 1 Pseudo-code for Volume Bar Construction

Set the volume threshold V^*
Initialise $B \leftarrow \emptyset$ ▷ Collection of completed volume bars
Initialise $v \leftarrow 0$ ▷ Cumulative traded volume
Initialise $S \leftarrow \emptyset$ ▷ Current slice of order book records
for each record r in the order book **do**
    if r corresponds to an executed trade **then**
        $v \leftarrow v + \text{size}(r)$
    end if
    Append r to S
    if $v \geq V^*$ **then**
        Construct bar features from S and append to B
        Reset $v \leftarrow 0$, $S \leftarrow \emptyset$
    end if
end for
return B

3.2.2 Label Construction

Let P_t be the executed close price of bar t, and define the close-to-close (net) return

$$r_t := \frac{P_t - P_{t-1}}{P_{t-1}}, \qquad t = 1, 2, \ldots,$$

with the first row (involving P_0) dropped. To make labels regime-aware, we follow common practice (e.g., López de Prado (2018, Ch. 3 p. 44)) and use an exponentially weighted moving standard deviation (EWMSD). In recursive form, for a decay parameter $\alpha \in (0, 1)$, we define the exponentially weighted mean μ_t and variance v_t via

$$\mu_t \;=\; \mu_{t-1} + \alpha\,(r_t - \mu_{t-1}), \qquad v_t \;=\; (1-\alpha)\,v_{t-1} + \alpha\,(r_t - \mu_{t-1})^2,$$

with initialisation $\mu_1 = r_1$ and $v_1 = 0$. The EWMSD is $\sigma_t := \sqrt{v_t}$.

We set $\alpha = 0.5$ to remain fairly agnostic and to capture regime shifts quickly without overreacting to noise, and we require a minimum of 100 samples before assigning labels to mitigate noisier initial estimates. Labels use the per-bar threshold σ_t:

$$y_t \;=\; \begin{cases} 0, & \text{if } r_{t+1} \leq -\sigma_t, \\ 1, & \text{if } |r_{t+1}| < \sigma_t, \\ 2, & \text{if } r_{t+1} \geq \sigma_t, \end{cases} \qquad \text{for } t \geq 100. \tag{3.1}$$

We will occasionally refer to the classes "down," "flat," and "up" by their numerical labels. This makes the targets scale with the prevailing volatility: in higher-volatility regimes, small returns are treated as less attractive, aligning with the idea that greater risk warrants higher expected compensation.

As seen in (3.1), we avoid data leakage by computing all moving statistics using only the current and previous bars, leaving the next-period return column (shifted back one step) untouched.

3.2.3 Feature Construction & Motivation

In the next subsection, we present the features selected for our models and the rationale for their inclusion. The feature construction is heavily inspired by López de Prado (2018).

3.2.3.1 Entropy Rate of Aggressor Direction

Shannon entropy provides a quantitative measure of uncertainty. For a discrete random variable X with support A,

$$H(X) = -\sum_{x \in A} \mathbb{P}(X = x)\, \log_2 \mathbb{P}(X = x).$$

This is maximised by the uniform distribution—i.e., when $\mathbb{P}(X = i) = 1/|A|$ for all $i \in A$, where $|A|$ is the cardinality of A. Intuitively, entropy is the expected number of bits required to inform the outcome of a random draw.

The entropy *rate* generalises this idea to a stochastic process. For $Z = \{Z_t\}_{t \in \mathbb{N}}$,

$$H(Z) := \lim_{n \to \infty} \frac{1}{n} H(Z_1, \ldots, Z_n),$$

where $H(Z_1, \ldots, Z_n)$ is the joint entropy. Following López de Prado (2018, Chap. 18, p. 263), we estimate the entropy rate of the liquidity-taker's direction sequence via its compressibility using the method of Gao et al. (2008). The method involves computing the length of previously seen repeated subsequences from a reference point in the series: short repeats imply high entropy, whereas long repeats indicate lower entropy.

Let $\mathbf{X} = (X_0, X_1, \ldots)$ be a sequence of discrete observations (we index from 0 to match Gao et al. (2008)). For each position $i \geq 1$ and $n \geq 1$, define

$$L_i^n(\mathbf{X}) := 1 + \max \Big\{ l \;\Big|\; X_i^{i+l-1} := (X_i, X_{i+1}, \ldots, X_{i+l-1}) \\ \text{is a sub-block of } X_{i-n}^{i-1} := (X_{i-n}, X_{i-n+1}, \ldots, X_{i-1}),\ l \in [1, n] \Big\}.$$

Thus, $L_i^n(\mathbf{X}) - 1$ is the length of the longest block starting at i that has already occurred among the previous n observations.

Ornstein and Weiss (1993) show that, almost surely,

$$\lim_{n \to \infty} \frac{L_i^n(\mathbf{X})}{\log_2 n} = \frac{1}{H(\mathbf{X})},$$

so this recurrence statistic yields an estimator of the entropy rate. Gao et al. (2008)

propose

$$\bar{H}_n(\mathbf{X}) := \frac{1}{n} \sum_{i=2}^{n} \frac{\log_2 i}{L_i^i},$$

and prove that, under stationarity, ergodicity, and a finite sequence, $\bar{H}_n(\mathbf{X})$ is a consistent estimator of $H(\mathbf{X})$; that is, for any $\varepsilon > 0$ there exists N such that for all $n > N$,

$$\mathbb{P}\big(|\bar{H}_n - H| \geq \varepsilon\big) \leq \varepsilon.$$

In our application, we encode aggressor direction using the standard LOBSTER convention: -1 for sell-initiated and $+1$ for buy-initiated trades, as discussed in Section 3.1. As a simplifying assumption, we assume the aggressor-direction sequence to be stationary and ergodic. The first trade in each volume bucket represents X_0, and subsequent trades extend $\mathbf{X}$. We compute $\bar{H}_n(\mathbf{X})$ up to $n = \lfloor N/2 \rfloor$, where N is the number of executed trades in the bucket.

Interpretation is straightforward: higher entropy in the aggressor sequence suggests greater market efficiency, with little redundancy or predictable structure in order flow; lower entropy indicates more regularity or predictability, which a model can learn to exploit alongside other features.

3.2.3.2 High-Low Volatility Estimator

Next, we construct an intra-bar, range-based volatility feature, motivated by its lower-variance properties (albeit with a slight bias; see Section 2.4). Concretely, we form a rolling high–low estimator over the past 100 volume bars, scaled by the Parkinson constant in $\sigma^2_{\text{Parkinson}}$ (equation (2.1)). This yields a high-frequency analogue that captures information in price ranges while preserving sufficient data for our models—because 100 bars are lost anyway due to label construction (Section 3.2.2), this window does not further reduce the effective sample.

We compute this measure separately on the bid and ask sides of the book, recognising that side-specific volatility can differ and that a single mid-price range may obscure these asymmetries. Including a volatility metric is standard in financial machine learning: our metric offers the model a lower-variance proxy for latent uncertainty and could act as an interaction partner for directional features.

3.2.3.3 Standard microstructure and financial features

Beyond the feature set described above, we add standard market microstructure and financial machine learning signals that capture order flow, depth, and execution conditions within each volume bar. All subscripts t refer to volume time.

Table 3.1: Summary of the 16 engineered features, listed in the order they are fed to the models. Feature IDs (1–16) are used for cross-referencing in later sections.

ID	Feature	Description
1	Final best bid size	Level 1 bid depth at the end of the bar.
2	Final best ask size	Level 1 ask depth at the end of the bar.
3	Hidden order proportion	Proportion of executed volume attributable to hidden liquidity within the bar.
4	Arrivals per second (bid)	Rate of order *arrivals* on the bid side within the bar.
5	Arrivals per second (ask)	Rate of order *arrivals* on the ask side within the bar.
6	Cancellations per second (bid)	Rate of order *cancellations* on the bid side within the bar.
7	Cancellations per second (ask)	Rate of order *cancellations* on the ask side within the bar.
8	Executions per second (ask)	Rate of order *executions* against the ask within the bar.
9	Executions per second (bid)	Rate of order *executions* against the bid within the bar.
10	Aggressor entropy rate	Entropy rate of the trade-initiator (aggressor) sequence within the bar.
11	Signed order flow	Normalised order flow in $[-1, 1]$, consistent with the LOBSTER sign convention.
12	Return (executed price)	Bar-to-bar return computed from executed trade price.
13	Return (VWAP)	Bar-to-bar return computed from VWAP.
14	Signed order flow AC1	Lag-1 autocorrelation of the signed order flow sequence.
15	High–low volatility (bid)	Parkinson's range estimator computed on the bid side.
16	High–low volatility (ask)	Parkinson's range estimator computed on the ask side.

VWAP and Execution Price Return. We include returns computed on the *volume-weighted average price (VWAP)* and on the *executed price*. VWAP, given by

$$\mathrm{VWAP}_t = \frac{\sum_{i=1}^{n} p_i |q_i|}{\sum_{i=1}^{n} |q_i|},$$

where p_i and q_i are the trade price and size of the i-th trade in a bucket, is widely used as a benchmark execution price. This captures a consensus price within the volume bar, while executed-price returns provide complementary information about price dynamics driven by actual transactions, which the model can learn to differentiate.

Event intensities (per-second rates). Let $\Delta t_t = \tau_t^{\mathrm{end}} - \tau_t^{\mathrm{start}}$ be the bar duration. We count, on each side of the book, the number of (i) limit orders placed, (ii) cancellations (partial or full), and (iii) executions: $N_t^{\mathrm{lim},s}$, $N_t^{\mathrm{can},s}$, $N_t^{\mathrm{exe},s}$ for $s \in \{\mathrm{bid}, \mathrm{ask}\}$. We

normalise by duration to obtain intensities

$$\lambda_t^{\text{lim},s} = \frac{N_t^{\text{lim},s}}{\Delta t_t}, \quad \lambda_t^{\text{can},s} = \frac{N_t^{\text{can},s}}{\Delta t_t}, \quad \lambda_t^{\text{exe},s} = \frac{N_t^{\text{exe},s}}{\Delta t_t}.$$

In the rare case of a single large order occupying an entire bar (four instances in our sample), we set all rate variables to 0 and set a direction indicator for the executed side to 1. Although this introduces a discontinuity in the true feature values, in combination with other features the model can learn that this corresponds to a bar dominated by one large order. Assigning 0 to these "missing" rate values is a legitimate way to treat missing values; regardless, since it affects only four samples out of 35,555 (all in the training set), it should have negligible impact.

Signed Order Flow. We also use the *normalised signed order flow*, defined as

$$\text{SOF}_t = \frac{V_t^+ - V_t^-}{V_t^+ + V_t^-},$$

where V_t^+ and V_t^- are the absolute buy- and sell-initiated volumes in the bucket, respectively. This feature, taking values in $[-1, 1]$, reflects whether trading pressure is dominated by buyers or sellers, and can help predict short-term price direction. In the spirit of López de Prado (2018), we also include the AC1 term, calculated on a rolling basis with a lookback of 100 windows, allowing the model to exploit immediate autocorrelation in this feature, while SOF supplies the pressure's sign.

Level 1 Depth. The *final best bid/ask size* records the depth available at the top of the order book at the end of the bucket. It measures immediate supply and demand near the best quotes, which can provide predictive content for short-term resilience or vulnerability—e.g., a much smaller best bid size than best ask size may indicate downward pressure.

Hidden to Visible Proportion. Finally, we consider the proportion of *hidden versus visible orders* executed within each bucket. Hidden orders can signal the presence of large, possibly informed traders concealing their full interest. They also imply fewer visible quote changes; the model may learn that such phases are associated with a higher probability of no price change, as herding behaviour is less likely when trades are hidden.

Together, these features capture complementary aspects of liquidity, order-flow imbalance, market depth, and trading behaviour, which we hypothesise could carry predictive information about short-horizon price direction movement. Notably, the rolling windows in both signed-order-flow AC1 and our intra-bar volatility proxies align with the 100-bar warm-up used in the label calculation; therefore, no additional data are lost.

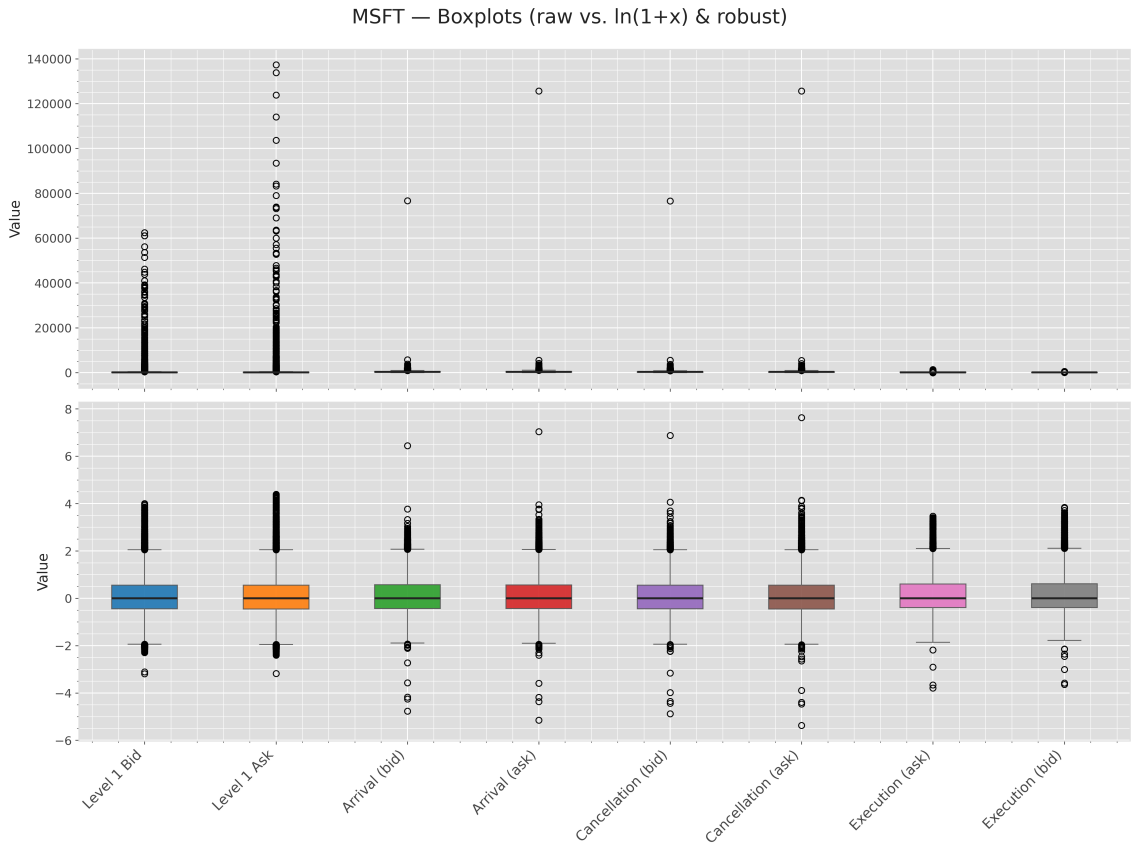


Figure 3.2: Boxplots of feature distributions before (top) any scaling and after (bottom) applying a $\ln(1+x)$ transform followed by robust standardisation (median/IQR). The transformation markedly attenuates extremes—previously several orders of magnitude above the median—yielding tighter, more comparable ranges. The largest and more common pre-transform spikes occur in MSFT's Level 1 depth features, where the large values are likely due to the stock's high liquidity.

3.2.4 Feature Normalisation

Because neural networks typically train more reliably when inputs are roughly bounded and comparable in scale, we attenuate a handful of extreme values—arising primarily from occasional very large Level 1 depth queue sizes—via a two-step transform. First, as our problematic features are non-negative and right-skewed, we apply a $\ln(1+x)$ transform. Second, we standardise using a `RobustScaler`,

$$x^\star = \frac{x - \text{median}(x)}{\text{IQR}(x)}, \qquad \text{where IQR}(x) = Q_{0.75}(x) - Q_{0.25}(x),$$

and $Q_p(x)$ denotes the $100p$th quantile of x. This centres each feature at its median and scales by its interquartile range, leveraging robust statistics to limit the influence of outliers. As Figure 3.2 illustrates, this procedure substantially tightens the empirical distributions: we do not obtain values exactly confined to $[-1, 1]$, but they are well centred around that band and the remaining extremes are far less pronounced. Importantly, all

of our neural networks train stably under this preprocessing, so we retain it as is. For additional truncation one could winsorise to the middle 98% (e.g., clamp to the $[1\%, 99\%]$ quantiles), but given the current stability and performance we opt not to introduce that extra intervention.

3.3 Rough Transformer

3.3.1 Signatures

Let $\mathcal{I}_d$ denote the set of all finite d-multi-indices, which we can define with d-tuples:

$$\mathcal{I}_d := \bigcup_{p\geq 1} \{1, \dots, d\}^p, \qquad \mathcal{I}_d^n := \{ I \in \mathcal{I}_d : |I| = n \},$$

where we conventionally define A^p, for a set A, to be its p-ary Cartesian power.

Let $X : [0, \infty) \to \mathbb{R}^d$ be a bounded and 1-time continuously differentiable function and let $0 \leq s \leq t$. For a multi-index $I = (i_1, \dots, i_p) \in \mathcal{I}_d^p$, the level-$p$ iterated integral of X over $[s, t]$ is, with abuse of notation:

$$S_{s,t}^I(X) := \int_{s<u_1<\cdots<u_p<t} dX_{u_1}^{i_1} \cdots dX_{u_p}^{i_p}, \tag{3.2}$$

where $dX_{u_j}^{i_p}$ denotes the differential of the i_p–th feature evaluated at u_j.

Collecting all multi-indices of cardinality n yields the level-n signature vector

$$\tilde{S}_{s,t}^n(X) := \big(S_{s,t}^I(X)\big)_{I\in\mathcal{I}_d^n} \in \mathbb{R}^{d^n}.$$

The full signature is the infinite sequence of levels

$$S_{s,t}(X) := \big(\tilde{S}_{s,t}^0, \tilde{S}_{s,t}^1(X), \tilde{S}_{s,t}^2(X), \dots\big), \qquad \tilde{S}_{s,t}^0 := 1,$$

and its truncation at level n is

$$S_{s,t}^{\leq n}(X) := \big(\tilde{S}_{s,t}^0, \tilde{S}_{s,t}^1(X), \dots, \tilde{S}_{s,t}^n(X)\big),$$

which has total dimension $\sum_{k=0}^n d^k = (d^{n+1}-1)/(d-1)$. In the Rough Transformer implementation, we omit the $\tilde{S}_{s,t}^0(\widehat{X})$ term, and begin our sequence with the level 1 signatures.

3.3.2 Transformer Extension

Our implementation closely follows Moreno-Pino et al. (2025). In this subsection, we first introduce an encoder-only Transformer in a time series setting (which we also use as a baseline), then extend to interpolation under irregular sampling, and finally detail the

rough-signature extension.

3.3.2.1 Time Series Encoder-Only Transformer

For time series data, we treat the input as a length-L sequence with d features per step, i.e., a matrix $\mathbf{X} \in \mathbb{R}^{L \times d}$. We first apply a learned linear projection

$$T : \mathbb{R}^{L \times d} \to \mathbb{R}^{L \times d_{\text{model}}},$$

lifting features to the model dimension. We then add a positional encoding $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{L \times d_{\text{model}}}$, using the standard sinusoidal scheme of Vaswani et al. (2023). Let

$$\mathbf{Z} = T(\mathbf{X}) + \mathbf{E}_{\text{pos}}.$$

Self-attention operates on $\mathbf{Z}$. For head $h = 1, \ldots, H$,

$$Q_h = \mathbf{Z} W_h^Q, \qquad K_h = \mathbf{Z} W_h^K, \qquad V_h = \mathbf{Z} W_h^V, \qquad W_h^{Q,K,V} \in \mathbb{R}^{d_{\text{model}} \times d_k},$$

and the (row-wise) scaled dot-product attention is

$$\text{Attn}_h = \text{Attn}(Q_h, K_h, V_h) = \text{softmax}\left(\frac{Q_h K_h^\top}{\sqrt{d_k}} \right) V_h.$$

Heads are concatenated and projected:

$$\text{MultiHead}(Q, K, V) = \left(\text{Attn}_1 \mid \cdots \mid \text{Attn}_H \right) W^O, \qquad W^O \in \mathbb{R}^{H d_k \times d_{\text{model}}},$$

where $(A \,|\, B)$ denotes horizontal concatenation for matrices A, B. The resulting embeddings pass through the MLP sub-layer, with residual connections and normalisation around both sub-layers (see Section 2.1).

3.3.2.2 Path reconstruction

We observe an irregular time series $\mathbf{X} = \{(t_j, X_j)\}_{j=0}^{L}$ with $0 = t_0 < \cdots < t_L$ and $X_j \in \mathbb{R}^d$. We construct a piecewise–linear path $\widehat{X} \colon [t_0, t_L] \to \mathbb{R}^d$ by linearly interpolating between consecutive samples:

$$\widehat{X}(t) \;:=\; \frac{t_{j+1} - t}{t_{j+1} - t_j} X_j \;+\; \frac{t - t_j}{t_{j+1} - t_j} X_{j+1}, \qquad t \in [t_j, t_{j+1}],\ j = 0, \ldots, L-1,$$

so that $\widehat{X}(t_j) = X_j$ and $\widehat{X}$ is continuous and of bounded variation (hence all iterated integrals in the signature are classically well-defined). All signatures below are computed on $\widehat{X}$.

3.3.2.3 Global–local (multi-view) signatures

Fix observation times $\{t_k\}_{k=1}^{\bar{L}} \subseteq \{t_j\}$. Define the multi-view signature at t_k by concatenating a global and a local summary:

$$M^{\leq n}(\mathbf{X})_k := \Big(S^{\leq n}_{t_0,t_k}(\widehat{X}) \,,\; S^{\leq n}_{t_{k-1},t_k}(\widehat{X}) \Big), \qquad k = 1, \ldots, \bar{L}, \tag{3.3}$$

where we omit the level-0 component $\tilde{S}^0_{s,t}(\widehat{X})$, as stated earlier. In our implementation, the global/local truncated signatures in (3.3) are treated as row vectors, and are appended to create another row vector. Stacking over k gives

$$M^{\leq n}(\mathbf{X}) := \big(M^{\leq n}(\mathbf{X})_1, \ldots, M^{\leq n}(\mathbf{X})_{\bar{L}} \big)^\top \in \mathbb{R}^{\bar{L} \times d_{\mathrm{sig}}}, \quad d_{\mathrm{sig}} \;=\; 2 \sum_{i=1}^{n} d^i.$$

Finally, $M^{\leq n}(\mathbf{X})$ replaces the standard per-step embedding $T(\mathbf{X})$ in the Transformer, i.e., the global and local signature calculations and concatenations replace the embedding space projection step. Rather than adding a positional embedding as before, we now *append* a learned p-dimensional positional vector to each token, forming

$$\mathbf{Z} \;=\; \big(M^{\leq n}(\mathbf{X}) \,|\, \mathrm{PE} \big) \;\in\; \mathbb{R}^{\bar{L} \times (d_{\mathrm{sig}} + p)},$$

where d_{sig} is defined as before and $\mathrm{PE} \in \mathbb{R}^{\bar{L} \times p}$ refers to the positional embedding matrix. This avoids destructive interference when d_{model} is small or varies (e.g., at low signature depth): in the original Transformer, addition is relatively safe because the high embedding dimension makes random vectors nearly orthogonal; here that is not guaranteed. By concatenating instead of summing, we preserve the signature coordinates intact while still injecting positional information as separate channels for the attention layers to exploit.

3.3.2.4 Univariate Framework

Moreno-Pino et al. (2025) also consider a univariate formulation that ignores cross–feature interactions: each feature is processed independently (with an appended time channel), and the resulting signature vectors are concatenated. Formally, one can form d *univariate* time-augmented vectors $\Xi^{(i)}(t) := (t,\, \widehat{X}^i_j(t))$, where $\widehat{X}^i_j$ is the i–th component of $\widehat{X}_j$. Then, compute signature features for each view up to depth n and concatenate:

$$M^{\leq n}_{\mathrm{uni}}(\mathbf{X})_k \;:=\; \Big(\underbrace{S^{\leq n}_{t_0,t_k}(\Xi^{(1)}), \ldots, S^{\leq n}_{t_0,t_k}(\Xi^{(d)})}_{\text{global}}, \underbrace{S^{\leq n}_{t_{k-1},t_k}(\Xi^{(1)}), \ldots, S^{\leq n}_{t_{k-1},t_k}(\Xi^{(d)})}_{\text{local}} \Big).$$

$M^{\leq n}_{\mathrm{uni}}(\mathbf{X})$ is obtained by stacking the per-time vectors as in the previous section, after which we append the positional embedding matrix.

3.3.2.5 Training paradigm

We begin with a two-dimensional array of 35,555 samples and 16 features, which we segment into fixed-length sequences and stack into a standard three-dimensional tensor (B, L, d), where B is the batch size, L the sequence length, and d the feature dimension. The sequence length is among the hyperparameters tuned.

Our tuning procedure involves running a *random search* with 50 trials, using early stopping with a patience of 10 epochs. For our Rough Transformer, the search ranges over both standard Transformer settings and Rough Transformer specifics—namely: sequence length L; the number of windows (selected points within each sequence, giving the reduced length $\bar{L}$); signature truncation depth n; and whether to compute univariate versus multivariate signatures. To mitigate validation overfitting and preserve a reliable estimate of generalisation error, we deliberately kept the search budget modest relative to the size of the parameter space. In addition, we restrict signature depth to $n \in \{2, 3\}$, since the feature space grows as $\mathcal{O}(d^n)$, making higher truncations memory-prohibitive. This restriction is not severe: empirical evidence suggests that modest depths ($n \leq 5$) are sufficient in practice (Moreno-Pino et al., 2025).

All tuning is performed on a 70%/15%/15% chronological train/validation/test split, preserving the temporal nature of the data. The training set is shuffled (with a fixed random seed), while validation and test sets remain unshuffled. The full hyperparameter search space and the selected best configuration for all our models are reported in Table A.1.

After selecting the best configuration, we retrain on the concatenated train and validation data using the exact tuned settings. To account for the larger dataset, we scale the stopping epoch proportionally to the increase in sample size: if $\hat{e}_\star$ is the validation-optimal epoch found on the train split, we set

$$e_{\text{final}} \;=\; \hat{e}_\star \times \frac{N_{\text{train}} + N_{\text{val}}}{N_{\text{train}}}.$$

This reflects the greater scope to train before overfitting when more data are available (cf. Chollet (2021)). Finally, we evaluate on the remaining 15% test set, reporting a confusion matrix and the mean $\pm$standard deviation of the standard metrics over 10 seeded runs.

3.3.2.6 Model-selection criterion

We select the final model by the lowest *validation cross entropy* (log-loss), rather than accuracy or some multi-label F1 score. Cross entropy rewards general well-calibrated confidence, even at the expense of accuracy, and penalises overconfident mistakes—exactly what we want with noisy, class-imbalanced LOB labels. Notably, we do not use an F1 score for model selection because our training often starts at a majority-class baseline, after which our model quickly overfits. Our macro F1 begin low and drift upwards to class-

proportion baselines (the models begins predicting randomly on the validation set). This provides an unstable selection signal. Accordingly, we simply report accuracy and macro F1 (to clearly delineate any failure of minority class prediction) as secondary diagnostics.

We also report per-class, one-vs-rest metrics—precision, recall, and the binary F1 for each class. Specifically, for class k, we collapse the three-way task into a binary task (k vs. not-k). These per-class scores provide a more granular view than aggregate three-class metrics: they reveal, for example, whether the model is particularly strong on a minority class, and they allow us to diagnose whether improvements arise from higher precision (being more often correct when predicting k), higher recall (capturing a larger share of true k), or both. The F1 score provides a conservative, single-number summary for deciding which model performs better on a given class. As the harmonic mean of precision and recall, it penalises low values in either component more strongly than an arithmetic mean, discouraging imbalanced trade-offs between the two.

The core code for our training pipeline is provided in Appendix B, including the data formatting (prior to being fed into the training set) and the training loop itself.

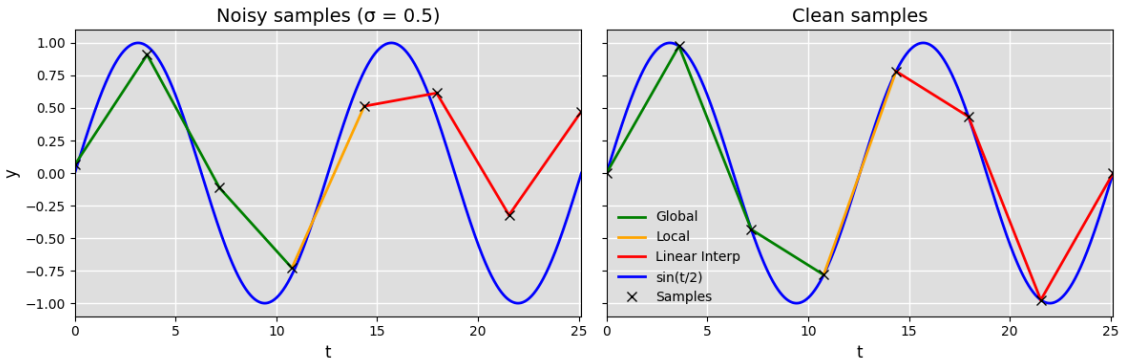


Figure 3.3: Signature computation on a single feature following $\sin(t/2)$. Left: Gaussian noise is added to the samples ($\sigma = 0.5$); right: noise–free. The yellow segment marks the local window used for the *local* signature from the 4th to the 5th point, while the green segment spans the *global* path used for the global signature (and includes the yellow segment). With sparse sampling or noise, the piecewise–linear interpolation (red) can deviate from the generating curve (blue), illustrating the risk of overfitting sampling noise or miscapturing the true distribution.

3.3.2.7 On the linear interpolation construction

Figure 3.3 highlights a potential limitation of the standard piecewise–linear lift: under sparse sampling or high-variance observations, the linear approximation to a smooth path, for example $\sin(t/2)$, can be crude, so derived signatures risk erroneously approximating a completely wrong feature-generating process. While the linear error contracts with denser sampling, in low-resolution or noisy settings a smoother lift (e.g., a smoothing spline) might be preferable before computing signatures. That said, perhaps the roughness introduced by piecewise–linear interpolation can act as implicit regularisation, potentially mitigating

overfitting in our low–signal, high–noise setting. It may also be preferable if the feature–generating process is genuinely jumpy and lacks smoothness.

Therefore, we implement a *smoothing spline* Rough Transformer simply to compare to the piecewise–linear variant, consistent with Moreno-Pino et al. (2025), who emphasise that piecewise–linear interpolation is often chosen for computational convenience. For tractability, we adopt the default smoothing factor used by `SciPy`,

$$s \;=\; m - \sqrt{2m},$$

where m is the number of samples in the series (here $m = L$: the sequence length hyperparameter). Because efficient libraries for signatures of non–piecewise–linear paths are not widely available, we use a pragmatic surrogate: subdivide each interval into $r = 8$ evenly spaced subsamples and compute the fast piecewise–linear signature with `iisignature`, as before. This refinement closely approximates the smoothing spline lift at much lower cost. Crucially, this does not defeat the purpose of the spline: the surrogate signature is computed on a grid that is much finer than the points used by the standard Rough Transformer, thereby preserving the smoothing benefits while remaining computationally efficient.

As noted, we refer to this variant as the *Spline Transformer*, while the original piecewise–linear version remains the *Rough Transformer*.

3.4 Baselines

Vanilla Transformer. A standard encoder-only Transformer operating on the same (B, L, d) tensors serves as an *architectural control*. It isolates the marginal benefit of replacing raw token embeddings with signature-based embeddings in the Rough Transformer.

Recurrent Dropout LSTM. The use of an LSTM offers an inductive bias complementary to a Transformer—both are temporal models—allowing a direct comparison between two distinct time series architectures. Notably, we implement our own variant, incorporating recurrent dropout for additional regularisation in this high-parameter setting to help mitigate overfitting. This further ensures a fair comparison with our main model, which has scope for intense regularisation.

Random Forest (RF). An ensemble extension of decision trees that combats overfitting by *bagging*: each tree is trained on a bootstrap sample of the training set, and at each split a random subset of features is considered. This decorrelates trees so that their errors do not align; final predictions are obtained by majority vote (classification) across the forest. In our setting, we collapse the 3D tensor into a matrix of dimensions $(B, L \times d)$

(simple aggregations of the same inputs). We chose to use an RF due to its ability to fight overfitting (conducive to low signal financial data), as well as its robustness to class imbalance.

NAIVE BASELINES. Let the training-set class proportions be $\widehat{\pi}_c = n_c/N$ for our labels $c \in \{0, 1, 2\}$. To simulate what is known at training time, all constants below are computed on the *train fold only* and then evaluated on validation/test.

Cross Entropy Baselines. We assume a constant-probability model $\mathbb{P}(y = c \mid x) \equiv \widehat{\pi}_c$. The cross entropy baseline is

$$\mathcal{L}_{\mathrm{CE}}^{\mathrm{naive}} := -\sum_{c=0}^{2} \widehat{\pi}_c \log \widehat{\pi}_c.$$

Accuracy Baseline. Let $c^\star = \arg\max_c \widehat{\pi}_c$. Predict $\widehat{y} \equiv c^\star$, giving an accuracy baseline of:

$$\mathrm{Acc}^{\mathrm{naive}} = \widehat{\pi}_{c^\star}.$$

These closed-form baselines provide clear sanity checks against which we compare all trained models. If our models fail to outperform these naive baselines, they are likely of limited value.

NOTE ON HYPERPARAMETER TUNING. All baselines use the same preprocessing, data splits, and optimisation protocol as the Rough Transformer; only their model-specific hyperparameters are differ (see Table A.1). Because the neural models (Transformer, Rough Transformer, LSTM) have high parameter counts relative to our dataset, they are prone to overfitting. We therefore bias the search toward *strong regularisation*: higher dropout in embedding/residual/attention blocks and tighter depth/width limits (fewer layers, a moderate d_{model}). This keeps effective capacity in check and helps mitigate overfitting from the outset.

3.5 ABLATION STUDY

To assess whether our models learn anything beyond chance, we conduct a label–permutation test following Ojala and Garriga (2009). The null hypothesis is that there is *no* class structure linking inputs to labels—i.e., any measured performance arises from chance alignment. We fix preprocessing, splits, and the tuned hyperparameters, then train each architecture (Rough Transformer, Spline Transformer, Vanilla Transformer, LSTM, RF) on the *true* labels across 10 independent seeds and record the observed statistic T_{obs} as the mean *test* cross entropy. While Ojala and Garriga (2009) report results with default hyperparameters, using our tuned settings does not alter the validity of the test: under the null (no feature–label alignment), a validation refined specification should not systematically

improve performance beyond validation overfitting.

A standard permutation test assumes *exchangeability*: the joint distribution of the sample is invariant under permutations. This is routinely violated in time series. For example, for a three-state Markov chain (S_t) with initial distribution $(0.5, 0.25, 0.25)$ and uniform transition probabilities,

$$\mathbb{P}(S_1 = 1, S_2 = 2) = 0.5 \times \tfrac{1}{3} \;\neq\; 0.25 \times \tfrac{1}{3} = \mathbb{P}(S_1 = 2, S_2 = 1),$$

so naively shuffling labels could violate such assumptions. As a pragmatic, dependence-aware remedy, we construct the null by *cyclically shifting* the training labels (cf. Kingsbury et al. (2019)): for each replicate we draw a shift k uniformly from $\{L, \ldots, N - L\}$ and rotate

$$(y_1, \ldots, y_N) \;\mapsto\; (y_{1+k}, \ldots, y_N, y_1, \ldots, y_k),$$

where N is the (train+validation) length and L is the sequence length. Choosing $k \geq L$ ensures that none of the same features are used at any point in time to predict that label. This preserves class counts and retains some shorter-range temporal dependence while still breaking pointwise feature–label alignment.

We then form a Monte Carlo null distribution by repeating, for $b = 1, \ldots, B$ with $B = 100$, the following: apply a fresh cyclic shift to the train and validation labels, retrain the model with identical settings (using a fixed seed schedule), and evaluate the cross entropy loss on the permuted test set to obtain T_b. The one-sided p-value for "better than chance" (smaller cross entropy) is

$$p \;=\; \frac{1 + \sum_{b=1}^{B} \mathbf{1}_{\{T_b \leq T_{\text{obs}}\}}}{B + 1} \qquad (B = 100),$$

where $\mathbf{1}_{\{x \in A\}}$ is the indicator function. Our choice of 10 seeds and $B = 100$ follows Ojala and Garriga (2009), who note these settings are sufficient in practice; with our large dataset, the resulting test also has high power.

As we will show, SHAP attributions for our best–performing model (the LSTM) are most pronounced on a single feature. To quantify its contribution, we apply the test described above to the LSTM with that feature removed, as well as the original five models, as mentioned previously. Because we are testing multiple hypotheses simultaneously, we control the false discovery rate (FDR) using the procedure in (Benjamini and Hochberg, 1995) at level $\alpha = 0.05$.

Let $m = 6$ and let $p_{(1)} \leq p_{(2)} \leq \ldots \leq p_{(6)}$ denote the sorted p-values of our models. Define

$$\ell \;=\; \max\Big\{\, i \in \{1, \ldots, m\} \;:\; p_{(i)} \leq \tfrac{i}{m}\,\alpha \Big\}.$$

If such an ℓ exists, we reject the null hypotheses corresponding to $p_{(1)}, \ldots, p_{(\ell)}$ (i.e., those

models are deemed to detect class structure beyond chance at FDR level $\alpha = 0.05$); otherwise, no nulls are rejected.

Ojala and Garriga (2009) also propose a complementary hypothesis test that permutes *features within each class*, as we discussed in Section 2.2. This targets a different null—namely, that predictions do not rely on feature dependencies—and is typically applied after the label–permutation test has rejected the "no signal" null. In our time–series setting, however, feature–permutation is not straightforward: as stated, the usual exchangeability assumption is generally violated. Conditioning on class and then shuffling features within class destroys all temporal ordering and short–range dependence; even a naive adaptation that cyclically shifts features within class (analogous to our label shifts) would produce windowed tensors with little-to-no intra-sequence temporal structure, undermining the test's validity. We propose a solution that leverages interactions among signature features; however, it is best understood after an exposition of our use of SHAP values. We defer its discussion to Section 3.6.2.

3.6 SHAP & Interpretability

3.6.1 Initial Usage

Lundberg and Lee (2017) explain a prediction of f at x with an *additive feature–attribution* surrogate g defined on a simplified binary input $z' \in \{0,1\}^d$ that indicates the presence (1) or absence (0) of each feature. The surrogate takes the form

$$g(z') \;=\; \phi_0 \;+\; \sum_{i=1}^{d} \phi_i \, z'_i,$$

where ϕ_0 is a baseline and ϕ_i quantifies the contribution of feature i near x. Local frameworks try to ensure g approximates f through a data-dependent mapping $h_x(x') = x$ from simplified to original inputs, i.e. $g(z') \approx f\big(h_x(z')\big)$ when $x' \approx z'$. Because SHAP is a *local* method, we can equivalently reason directly in terms of the true input–output pair (x, f) without explicit reference to z' and h_x.

As discussed in Section 2.3, SHAP values are a unique solution to the class with the previously mentioned desired properties. Specifically, the attribution for feature i is

$$\phi_i = \sum_{S \subseteq \{1,\dots,d\} \setminus \{i\}} \frac{|S|!\,(d - |S| - 1)!}{d!} \Big(v_x(S \cup \{i\}) - v_x(S) \Big), \qquad v_x(S) := \mathbb{E}\big[f(X) \mid x_S\big],$$

where x_S refers to the feature vector with features in S.

A convenient visualisation is the SHAP *beeswarm* for a chosen target (here, the logit of the best performing minority class). Features are ordered by global importance $\mathbb{E}[|\phi_i|]$ or |SHAP|. Each point shows a sample–level attribution $\phi_i(x)$: positive values push the

logit up (greater confidence in the class), negative values push it down. Colour encodes the raw feature value (blue = low, red = high), revealing patterns such as "high value ⇒ positive contribution."

Because our inputs are time–series tensors of shape (L, d) per sample (time × features), attributions index as $\phi_{t,i}(x)$. We summarise them with three complementary views: (i) a feature–level beeswarm after aggregating over time, $\tilde{\phi}_i(x) = \sum_{t=1}^{L} \phi_{t,i}(x)$, ranked by $\mathbb{E}[|\tilde{\phi}_i|]$ for a time–agnostic overview; (ii) a *time profile*, $m_t := \mathbb{E}_{\text{batch}}\big[\frac{1}{d}\sum_{i=1}^{d} |\phi_{t,i}(x)|\big]$, indicating when the model is most sensitive; and (iii) a *feature–time heatmap* with entries $H_{t,i} := \mathbb{E}_{\text{batch}}[|\phi_{t,i}(x)|]$ to expose influential features in time.

We keep SHAP on the *logit* scale (rather than probabilities) to preserve strict local additivity. Now, because our models are differentiable and exact SHAP computation is costly, we use *GradientSHAP*.

3.6.2 Feature Interaction Detection

Because standard SHAP values quantify primarily univariate contributions, and given the constraints outlined in Section 3.5, we developed a procedure to highlight promising pairs of interacting features.

As is displayed in the Table A.1, our best Rough Transformer uses *univariate* signatures with depth $n = 2$; however, to probe the interaction effects naturally induced by signature features, we also analyse a slightly worse-performing variant with *multivariate* attention and signature depth $n = 2$. This allows us to identify which feature interactions drive predictions in that setting.

Concretely, we compute SHAP values for the multivariate Rough Transformer at the classifier input, aggregate them time-wise to obtain a batch-by-feature matrix, and then render a standard SHAP beeswarm plot to highlight signatures with the largest absolute contribution to the logits, as we have outline in the previous section. We provide a mapping from each column of the multi-view attention representation to: (i) term type (global vs. local), (ii) signature level k, and (iii) the involved channel(s)—singletons for $k = 1$, ordered pairs for $k = 2$—with channel IDs aligned to Table 3.1. Finally, this mapping allows us to nominate promising feature pairs: we select five top-ranked pairs and plot representative SHAP interaction dependence plots, thereby avoiding exhaustive enumeration of the $\binom{16}{2} = 120$ possible combinations while still investigating potentially informative interactions.

This procedure preserves temporal dependencies and offers a clear readout of which interaction features our best model can exploit. Its main limitation is that, with $n = 2$, it only captures pairwise interactions, and the multivariate Rough Transformer might not recover the same relationships as our best model leverages.

Chapter 4

Results

4.1 Main Results

4.1.1 Training Dynamics

Figure 4.1 displays training and validation accuracy/loss for the LSTM, Rough Transformer, Spline Transformer, and Vanilla Transformer. All models begin near the naive baselines; subsequently, model confidence improves markedly—validation cross entropy falls well below the naive loss—yet not enough to shift the largest logit away from the majority class. Consequently, validation accuracy remains close to its baselines and improves negligibly. Overfitting begins early, as evidenced by the widening train–validation gap. The lowest *validation* cross entropy is reached at epochs 8 (LSTM), 9 (Rough Transformer), 5 (Spline Transformer), and 17 (Vanilla Transformer). Beyond these points, loss divergence is apparent for all but the Vanilla Transformer, whose train/validation accuracy changes only marginally thereafter. Notably, this pattern persists even though all architectures consistently favour simpler designs, shorter input horizons and strong regularisation. In particular, the best results arise when using only the previous five buckets, and all signature-based models select a univariate specification. For reproducibility, exact hyperparameters are listed in Appendix A.1.

Finally, the models trained on raw (non-signature) features—the LSTM and Vanilla Transformer—exhibit better early gains in validation loss and, in our runs, achieve lower cross entropy than the signature-based variants, reinforcing the preference for simpler representations.

4.1.2 Test Set Results

Table 4.1 summarises test performance over 10 seeded runs: we report means ± standard deviations for all metrics and 95% confidence intervals (CIs) for cross entropy loss. The LSTM attains the lowest mean loss (0.874) and the highest mean macro F1 (30.15%), while

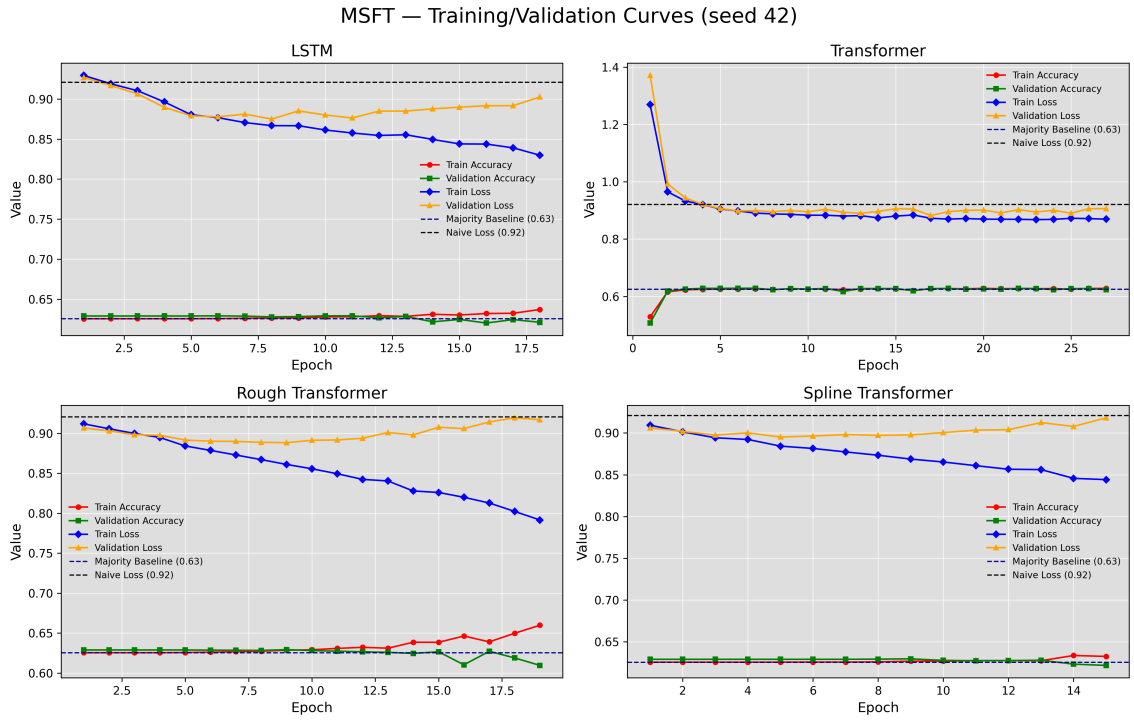


Figure 4.1: Training–validation accuracy and loss curves for the four models for which these diagnostics are available. Dashed horizontal lines mark the naive baselines, $\mathcal{L}_{\text{CE}}^{\text{naive}}$ and $\text{Acc}^{\text{naive}}$. Most models begin to overfit after only a few epochs despite substantial regularisation.

the Spline Transformer achieves the highest mean accuracy at finer precision (63.06%). The Vanilla Transformer is the most seed-sensitive, exhibiting the largest across-run loss standard deviation. Moreover, the 95% CIs for cross entropy are generally non-overlapping across models; the sole exception is the Rough Transformer vs. the Vanilla Transformer, whose endpoints appear to coincide at 0.882 (upper for Vanilla, lower for Rough) to **3 s.f.**; however, at higher precision, they do not.

All models beat the naive loss and accuracy baselines (0.921 and 62.54%, respectively), although the test majority-class proportion is 63.05%. A consistent pattern emerges: models with lower loss tend to achieve higher macro F1, yet often register lower accuracy than higher-loss models (the slight underperformance of the Transformer vs. Random Forest and Spline Transformer is masked by rounding). Echoing the validation-set behaviour, accuracy is notably "stubborn" across seeds—its variation is effectively zero at 2 d.p.—whereas loss does see some variation. In short, models, like in the validation set, become more confident (lower cross entropy) without generally shifting the predicted `argmax` away from the majority class; and in the cases where they do, the aggregate shifts generally yield lower accuracy than the higher-loss models.

As further evidence, Figure 4.2 presents the confusion matrices for all models on the first of the ten seeded runs. The Random Forest and Spline Transformer rarely predict anything other than the majority class; accordingly, their test accuracies lie close to the

Table 4.1: Model metrics on MSFT volume-sampled test data over 10 seeded runs, reported as mean ± standard deviation. cross entropy loss (with 95% confidence intervals) is rounded to **3 s.f.**; accuracy and macro F1 are percentages rounded to **2 d.p.** The best value in each column is **bold**. The LSTM attains the lowest loss and highest macro F1, while the Spline Transformer achieves the highest accuracy. Notably, where values appear tied at the displayed precision, the bold entry is strictly better at higher precision.

Model	CE Loss	95% CI (Loss)	Acc (%)	Macro F1 (%)
LSTM	$\mathbf{0.874 \pm 0.001}$	[0.873, 0.874]	62.93 ± 0.00	$\mathbf{30.15 \pm 0.00}$
Rough Transformer	0.882 ± 0.001	[0.882, 0.883]	62.80 ± 0.00	29.02 ± 0.00
Spline Transformer	0.888 ± 0.001	[0.888, 0.889]	$\mathbf{63.06 \pm 0.00}$	25.85 ± 0.00
Vanilla Transformer	0.880 ± 0.003	[0.878, 0.882]	63.06 ± 0.00	27.23 ± 0.00
Random Forest	0.888 ± 0.001	[0.887, 0.888]	63.06 ± 0.00	25.82 ± 0.00

test-set majority proportion. By contrast, the LSTM, Rough Transformer, and Vanilla Transformer produce substantially more varied predictions across classes. As shown in Table 4.2, these lower-loss models are consistently and comparatively stronger on the "up" class (class 2, see 3.2.2), achieving higher binary F1 scores primarily due to improved recall. For example, the LSTM attains similar precision for the "down" and "up" classes (32.14% vs. 32.31%), yet the F1 for the "up" class is roughly 25% higher than for "down," driven by higher recall (4.80% vs. 3.72%).

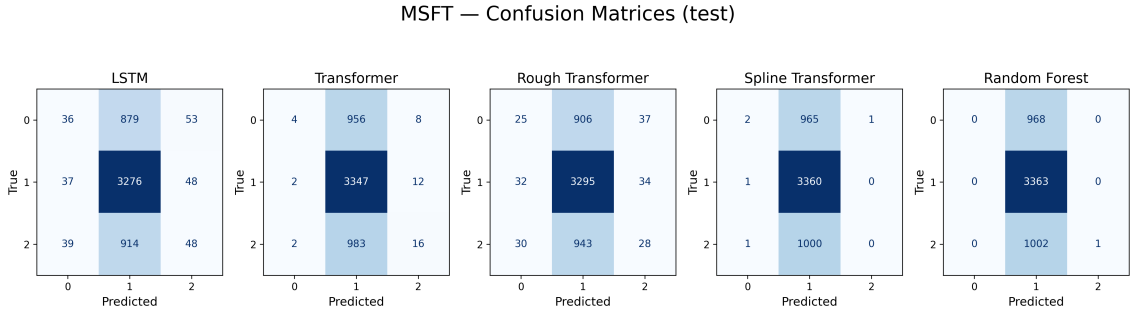


Figure 4.2: Confusion matrices for each model at its best-performing hyperparameters (seed = 42). The LSTM and Rough Transformer make the most minority-class predictions, trading off overall accuracy; by contrast, the Spline Transformer and Random Forest rarely predict minority classes, inflating accuracy but depressing recall on those classes.

4.2 Ablation Results

Table 4.3 reports the cyclic permutation label test across models. All p-values equal 0.01, i.e., at the resolution of 100 permutations none of the null losses exceeded any of the 10 true-run losses, allowing us to reject the null of no label–feature association at the 5% level (FDR-controlled). We also report the mean z-score across the 10 true runs (computed against each model's null-loss distribution) to quantify how far the true losses lie from the null mean in units of standard deviation. As shown, all full models lie more than three

standard deviations from their respective null means, consistent with the low p-values. However, when *Execution Return* is removed, the LSTM's z-score nearly halves. Notably, the Random Forest exhibits the largest effect at -26.28 standard deviations (lower is better for loss, hence the negative sign).

Table 4.2: Per-class Precision (**P**)/Recall (**R**)/**F1** for Classes 0–2 across all models. Values are percentages rounded to **2 d.p.** Comparing minority classes, lower-loss models attain higher binary F1 in class 2, indicating superior performance on the "up" class. All models tend to default to the majority class.

Model	**Class 0 (P/R/F1) (%)**	**Class 1 (P/R/F1) (%)**	**Class 2 (P/R/F1) (%)**
LSTM	32.14/03.72/06.67	64.63/97.47/77.72	32.21/04.80/08.35
Rough Transformer	08.74/02.58/04.74	64.06/98.04/77.48	28.28/02.80/05.09
Spline Transformer	50.00/00.21/00.41	63.10/99.97/77.37	00.00/00.00/00.00
Vanilla Transformer	50.00/00.41/00.82	63.32/99.58/77.41	44.44/01.60/03.09
Random Forest	00.00/00.00/00.00	63.06/100.00/77.35	100.00/00.01/00.20

4.3 Interpretability Results

Figures 4.3 summarise the interpretability outputs for our best model (LSTM), which is displayed in terms of the "up" class logits. Ranking variables by mean |SHAP| highlights *Execution Return* as the most informative; lower returns seem to increase model confidence, while higher returns reduce it. This relationship is, however, rather noisy. The next features show a clear model consensus across high–low counterparts; for instance, *Executions (bid)* behaves intuitively: when the number of bid-side trades is low, the model assigns greater confidence to an "up" movement. The time-importance plot yields another intuitive pattern—feature importance decays with lag—indicating that the most recent volume bars are most influential for prediction. Finally, the heatmap summarises these effects batch-wise and reveals that *Execution Return* remains most informative across the five volume bars.

Table 4.3: Label cyclic permutation test results for each model using 10 true runs and 100 permuted runs. We report (i) the p-value (with FDR control at level α) to assess rejection of the null, and (ii) the mean z-score across the 10 true runs to quantify how far the true loss departs from the null mean in terms of standard deviation. All models reject the null of no association between labels and features at $\alpha = 0.05$.

Model	**p-value**	**Average z-score**
LSTM	0.01	-4.37
Rough Transformer	0.01	-4.49
Spline Transformer	0.01	-3.50
Vanilla Transformer	0.01	-4.85
Random Forest	0.01	**-26.28**
LSTM–No Execution Return	0.01	-2.38

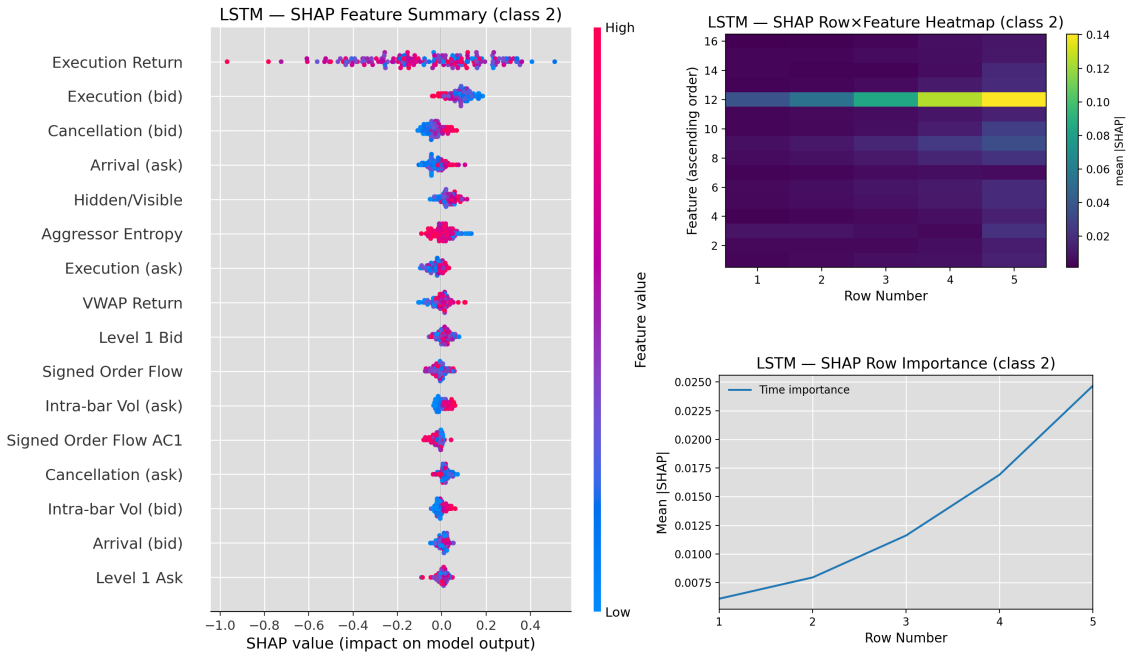


Figure 4.3: LSTM SHAP interpretability panels: (left) feature summary; (right) heatmap (top) and time importance (bottom). Feature summary plot is sorted from top down in terms of largest |SHAP| value; *Execution Return* is the most informative. Time plot indicates greater importance in more recent rows, while the heatmap aggregates this information batch-wise.

As mentioned in Section 3.6.2, given the $\binom{16}{2} = 120$ pairwise feature interactions under consideration, we leverage a different Rough Transformer's spatially aware signature processing and report the top ten *multivariate* SHAP interactions in Figure 4.5.[1] The panels reveal clear separation across features, with the five strongest (among distinct features) identified as $(5, 12)$, $(16, 15)$, $(12, 4)$, $(5, 7)$, and $(1, 13)$, spanning both global and local paths. Notably, two of these pairs involve our most informative feature, reinforcing its potentially central role in the model's decision process.

As discussed, Figure 4.4 presents the pairwise interaction plots. When both intra-bar volatilities are low (high), the model is less (more) confident in an "up" move; note that these volatilities are themselves highly correlated. A similar pattern holds for the *Cancellation (bid)–Execution (ask)* pair.

For interactions involving executions and returns, effects are subtler: a low frequency of ask-side limit orders together with a high bar return tends to *reduce* the probability of an "up" move, whereas low bar returns combined with a high frequency of ask arrivals tend to *increase* model confidence—albeit with considerable noise. These two features exhibit strong negative correlation. No clear structure is apparent in the remaining pairs.

[1] As noted earlier, this multivariate signature model underperforms its univariate counterpart.

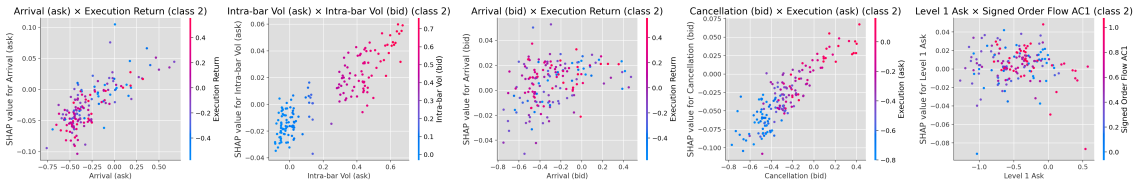


Figure 4.4: Interaction plots for the title-indicated feature pairs. We observe strong positive, near-linear associations among the intra-bar high–low volatility estimators, as well as between Cancellation (bid) and Execution (ask) rates; in both cases, SHAP values increase with the level of the corresponding variables. The Arrival (ask) × Execution Return panel is most interesting: despite noise, higher returns coupled with lower arrival rates are generally associated with lower SHAP values.

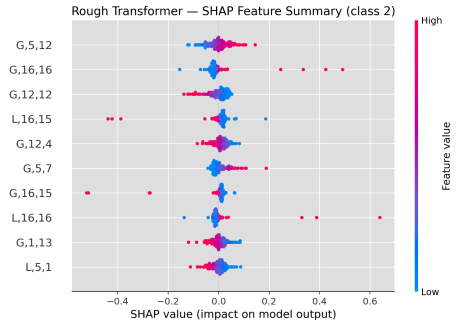


Figure 4.5: Beeswarm plot for the best-performing two-signature, multivariate Rough Transformer. "G" denotes global signatures, "L" denotes local, and the numbers map to feature IDs in Table 3.1. We use this beeswarm to identify which two-way signature interactions induce the largest confidence shifts, guiding our interaction-plot selections in Figure 4.4.

Chapter 5

Discussion

5.1 Research Question 2

We address **RQ2** first, as it is both easier to resolve than **RQ1** and provides context that guides the interpretation of **RQ1**.

Much like on the validation set, model loss varies little across models on the test set; indeed, the gap between models is slightly tighter on test. The results also suggest we did not markedly overfit to the validation set—unsurprising given strong regularisation, short training schedules, and a limited random search. A plausible explanation for the competitive test performance (compared to train baselines) is the slightly larger majority-class share in the test set (cf. 62.54% vs. 63.05%), which makes "always-majority" behaviour look better. Apparent improvements from validation to test therefore need not reflect learning; they are consistent with class-mixture differences.

This pattern is visible in the confusion matrices (Figure 4.2): models such as the Random Forest and the Spline Transformer make few predictions outside the majority class. The lone correct minority prediction for the Random Forest is plausibly due to chance, even if class 2 prediction is better than class 0 predictions across models. By contrast, the LSTM and the Rough Transformer do achieve some minority-class hits, albeit at a cost to accuracy. Notably, this accuracy drop coincides with lower cross entropy: on average, these models are more confident about true outcomes, but on the subset of instances where confidence is high enough to flip the `argmax` to a minority class, errors become more visible—hence lower loss yet slightly worse accuracy. This behaviour could suggest a *weak* noisy signal in the data.

The p-values in Table 4.3 support this interpretation, indicating extremely unlikely draws under the null of no class structure. Using the Benjamini–Hochberg procedure, we reject the null in all cases. This result corroborates that the models are learning something, lending plausibility to the presence of a weak signal.

Accordingly, we answer **RQ2**: there is likely class structure in our dataset.

5.2 Research Question 1

We have outlined in the previous section that test performance is rather modest: the immediate, high-level answer to RQ1 therefore appears to be **no**. Nonetheless, the results point towards a slightly more nuanced answer. Having established the presence of a weak signal in our data, we next consider which representations exploit it most effectively.

Non-signature methods consistently outperform signature-based approaches, plausibly because signature computation introduces compression and mild information loss, and because interpolation can misrepresent aspects of the underlying data-generating process. Within signature based methods, the Spline Transformer's underperformance could be explained with a feature-generating process containing frequent jumps, for which a smoothing spline is typically inferior. Generally though, the compression cost is likely less pronounced in our setting—models favoured very short sequence lengths and the chosen number of windows closely matched the true horizon—which helps explain the small cross-model loss differences among the signature vs. non-signature models.

The Rough Transformer, however, enjoyed lower z-scores than the LSTM and Vanilla Transformer, as seen in Table 4.3, indicating better relative performance to the null mean. However, this can simply be explained by larger null means compared to the non-signature temporal models, as well as a lower standard deviation, inflating the distance from the mean. This phenomenon is observed for the Random Forest, even though it has the worst loss performance.

Representation choice is not the only factor shaping performance. Among models that more readily predict outside the majority class (LSTM, Rough Transformer, Vanilla Transformer), class 2 is consistently predicted better than class 0, as seen in the higher per-class F1 scores in Table 4.2. While class 2 is only slightly more prevalent than class 0 among the minority classes, models allocate a similar number of predictions to both (see the comparable row counts in Figure 4.2). Thus, models are not simply defaulting to class 2 as a "next majority"; rather, the weak but genuine signal appears to manifest more strongly in class 2.

The performance gap arises because predictions for class 2 have proportionally higher recall, which is a harder achievement, as the model must shift the decision away from the majority default more times. This could help explain the lower cross entropy observed for these models: since class 2 is the second most common class, higher recall suggests the logits are generally being nudged in the correct direction for a larger share of the data; even if performance on class 0 slightly worsens the loss, the overall cross-entropy still improves.

Accordingly, our answer to **RQ1** is that, although the models do not manage to consistently predict the return direction, often performing worse than a majority accuracy would on a test set, the features do appear to inform class 2 movements more consistently

than class 0.

5.3 Research Question 3

Because our features appear to carry more predictive information for class 2, SHAP summaries provide a natural way to gauge their relative contributions and to inform **RQ3**.

In our best model, the executed-price return has the largest mean |SHAP|. Its marginal dependence indicates that the times when model confidence increases most are when the bar return is low, whereas its largest decreases occur when the bar return is high. The considerable scatter across observations suggests that this feature's usefulness could manifest more clearly through *interactions* with other variables rather than as a standalone predictor. However, Figure 4.5 conveys the same message via the interaction between executed return and ask-side limit-order placement: the SHAP contributions again exhibit a form of negative autocorrelation in Execution Return. The interaction adds little incremental value, consistent with the strong negative correlation observed between the variables. Intuitively, when ask arrivals are scarce within a bar, higher bar returns are more likely, as buy pressure faces less resting liquidity to overcome—therefore, the interaction is likely not informative beyond this mechanism.

We believe a possible explanation for the slightly improved test set loss, with decreasing accuracy and consistently better class 2 prediction, is attributable to a well-known microstructure regularity: at very short horizons, returns exhibit weak *negative* first-order autocorrelation (e.g., due to bid–ask bounce; see Cont (2001)). Therefore, we consider the most plausible mechanism to be a learned lag-1 sign reversal driven by the negative autocorrelation: the network exploits this tendency and then attempts to use additional cues to assess whether the reversal is strong enough to warrant switching from the majority class. This aligns with the observed pattern of improving cross entropy loss but little to no accuracy gain: the model is often confident in the correct direction, yet not always by enough to flip the `argmax`. Occasional flips likely reflect a mixture of weak, noisy signals from secondary features and spurious correlations that the model intermittently treats as informative. The small p-value in Table 4.3 corroborates that there is a weak but genuine signal, which the model exploits alongside bar return, rather than being driven solely by noise and Execution Return. However, as expected, the z-score drops sharply when the bar return is removed, supplying further evidence of its substantive contribution.

We provide an example of a possible effect another feature could have on class 2 logits. Figure 4.4 shows positive SHAP contributions associated with increases in Parkinson's volatility. Note that, in these data, bid- and ask-side volatilities are highly correlated and therefore carry much of the same information, though the information they represent can still be useful. One plausible mechanism is as follows. For (approximately) normal data, the *variance of the sample variance* scales with the square of the true volatility (e.g.,

$\mathrm{Var}(S^2) = 2\sigma^4/(n-1)$), so higher σ implies greater noise in close–close return–based estimates. A similar effect may operate here: volume time partially restores normality, and when true volatility is elevated, the close–close label estimator exhibits larger random fluctuations, whereas the lower-variance Parkinson estimator remains comparatively stable (see Section 2.4). Given the large training set, the model may learn that, during high-volatility periods, exceedance of the EWMSD threshold could be more likely because close–close draws are more variable than Parkinson volatilities; hence, mislabelled outcomes can cause the model to lean on Parkinson volatility: price fluctuates more than is captured by the close–close volatility. This would not truly improve predictability, however, because the effective risk would be underreported.

Generally, the other SHAP diagnostics support our hypothesis on Execution Return. The time plot (Figure 4.3) shows that the largest contributions concentrate in the most recent periods—consistent with a lag-1 microstructure signal—while the heatmap (Figure 4.3) indicates that bar return attribution dominates overall, and does so persistently across time. Taken together, these findings suggest that the model leverages the sign alternation in the return series, and the observed strong signals in Figure 4.5 could be due to strong correlation or happenstance.

Accordingly, our answer to **RQ3** is that the model's attributions are dominated by executed return, with predictions consistent with the well-known lag-1 negative autocorrelation effect; other features appear to play a secondary, largely noisy role, modulating this primary signal only marginally or randomly.

Chapter 6

Conclusion & Further Outlook

We transformed raw LOBSTER MSFT data into a bespoke, volume-sampled dataset in which each row corresponds to 10,000 shares traded in volume time. Building on López de Prado (2018), we engineered features from the event stream during bar construction and used them to address our research questions. Nonparametric tests indicate genuine structure in the data, which the models exploit more reliably for class 2 than for class 0—answering **RQ2**. Model attributions further show that *Execution Return* is the most informative feature; plausibly, the models leverage standard microstructure regularities (e.g., bid–ask bounce) to nudge logits in the right direction. However, weak or noisy signals in the remaining features appear to cap these gains, possibly explaining why reductions in cross entropy do not translate into higher accuracy. This informs **RQ3**. Taken together, the insights from **RQ2** and **RQ3** yield a nuanced view of **RQ1**: although overall performance on standard metrics remains modest, we observe more consistent improvements for class 2, which could be driven by Execution Return with additional—albeit weak—contributions from secondary features. This is supported by rejection of the null of no feature–label structure even for the best LSTM model with *Execution Return* removed.

Our study has several limitations that suggest concrete avenues for follow-up. First, many features are highly—though not perfectly—correlated. Such collinearity can cause variables to encode the same information, diluting apparent contributions in attribution summaries and adding dimensions that exacerbate overfitting without commensurate benefit. Future work should therefore investigate alternative (and ideally orthogonal) feature sets. This may include signals from upstream machine-learning tasks—such as news-based sentiment for the focal asset—or some nuanced insight from other, ideally under-exploited, datasets.

Moreover, for computational reasons, we fixed a single cumulative-volume setting on one highly liquid asset. While such assets provide abundant data and lower price impact, they are typically more informationally efficient and thus harder to exploit. Extending the analysis across assets spanning different liquidity tiers and sectors—and varying the

cumulative-volume horizon beyond 10,000 shares—would better gauge the scope for alpha across a broader range of assets and horizons; for example, longer horizons may allow more signal to materialise. In addition, a systematic comparison of volume time with transaction time and fixed calendar-time sampling would clarify differences among these paradigms and directly quantify any benefit from adopting volume time.

Similarly, computational constraints led us to fix the Spline Transformer's smoothing factor during hyperparameter tuning. With a larger budget, this parameter could be treated as a tunable hyperparameter, fitting multiple smoothing–spline lifts of the feature–generating process and assessing their impact on performance. We note, however, that any apparent gains may simply reflect validation overfitting. More broadly, this and other hyperparameters may be tuned under a more robust validation regime (e.g., purged K-fold cross-validation), while also using the triple-barrier labelling method to simulate stop-loss limits (López de Prado, 2018) and directly embed this trading practice in the labels.

Class imbalance is another constraint: the skewed label distribution makes majority-class predictions unduly attractive. Because misclassifying "flat" moves is typically less consequential than directional errors in an actionable setting, future work should explore cost-sensitive objectives (e.g., weighted cross entropy) with costs either pre-specified or learned. Such objectives will not create signal—if features are uninformative, performance will not improve—but they may better exploit genuine minority-class signal where it exists.

We did not implement class-conditional cyclic permutations for time-series data, owing to exchangeability concerns and the lack of a ready workaround. Investigating alternative within-class cyclic–shift schemes that break feature–label alignment while circumventing this issue would enable a natural continuation to the second hypothesis test of Ojala and Garriga (2009).

Finally, we reported SHAP values only for the best–performing model and examined dependence–interaction plots for just the top five feature pairs—selected from a different, underperforming model. Future work could instead apply Friedman's H-statistic to systematically screen for interacting pairs, then perform targeted ablations of the most influential features across *all* models after reassessing attributions. When testing the effect of removing supposedly informative features across multiple models, one could apply the Benjamini–Hochberg procedure to control the false discovery rate, as we have done.

Bibliography

Beckers, Stan. 1983. Variances of Security Price Returns Based on High, Low, and Closing Prices. *The Journal of Business* **56**(1) 97–112. URL https://www.jstor.org/stable/2352748. Publisher: University of Chicago Press.

Benjamini, Yoav, Yosef Hochberg. 1995. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological)* **57**(1) 289–300. URL https://www.jstor.org/stable/2346101. Publisher: [Royal Statistical Society, Oxford University Press].

Chen, Kuo-sai. 1958. Integration of paths—a faithful representation of paths by noncommutative formal power series. *Transactions of the American Mathematical Society* **89**(2) 395–407. doi:10.1090/S0002-9947-1958-0106258-0. URL https://www.ams.org/tran/1958-089-02/S0002-9947-1958-0106258-0/.

Chollet, François. 2021. *Deep Learning with Python, Second Edition*. 2nd ed. Manning Publications, Shelter Island, NY.

Cont, Rama. 2001. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance* **1**(2) 223–236. doi:10.1080/713665670.

Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function .

Easley, David, Marcos Lopez de Prado, Maureen O'Hara. 2012. The Volume Clock: Insights into the High Frequency Paradigm. doi:10.2139/ssrn.2034858. URL https://papers.ssrn.com/abstract=2034858.

Fama, Eugene F. 1970. Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance* **25**(2) 383–417. doi:10.2307/2325486. URL https://www.jstor.org/stable/2325486. Publisher: [American Finance Association, Wiley].

Gao, Y., I. Kontoyiannis, E. Bienenstock. 2008. Estimating the entropy of binary time series: Methodology, some theory and a simulation study. *Entropy* **10**(2) 71–99. doi:10.3390/entropy-e10020071. URL http://arxiv.org/abs/0802.4363. ArXiv:0802.4363 [cs].

Gould, Martin D., Mason A. Porter, Stacy Williams, Mark McDonald, Daniel J. Fenn, Sam D. Howison. 2013. Limit Order Books. doi:10.48550/arXiv.1012.0349. URL http://arxiv.org/abs/1012.0349. ArXiv:1012.0349 [q-fin].

Hornik, Kurt. 1991. Approximation Capabilities of Multilayer Feedforward Networks .

Kingsbury, Lyle, Shan Huang, Jun Wang, Ken Gu, Peyman Golshani, Ye Emily Wu, Weizhe Hong. 2019. Correlated Neural Activity and Encoding of Behavior across Brains of Socially Interacting Animals. *Cell* **178**(2) 429–446.e16. doi:10.1016/j.cell.2019.05.022. URL https://linkinghub.elsevier.com/retrieve/pii/S0092867419305501.

López de Prado, Marcos. 2018. *Advances in Financial Machine Learning*. John Wiley & Sons, Hoboken, NJ.

Lundberg, Scott, Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. doi:10.48550/arXiv.1705.07874. URL http://arxiv.org/abs/1705.07874. ArXiv:1705.07874 [cs].

Lyons, Terry. 2014. Rough paths, Signatures and the modelling of functions on streams. doi:10.48550/arXiv.1405.4537. URL http://arxiv.org/abs/1405.4537. ArXiv:1405.4537 [math].

Mandelbrot, Benoit, Howard M. Taylor. 1967. On the Distribution of Stock Price Differences. *Operations Research* **15**(6) 1057–1062. URL https://www.jstor.org/stable/168611. Publisher: INFORMS.

Moreno-Pino, Fernando, Álvaro Arroyo, Harrison Waldon, Xiaowen Dong, Álvaro Cartea. 2025. Rough Transformers: Lightweight and Continuous Time Series Modelling through Signature Patching. doi:10.48550/arXiv.2405.20799. URL http://arxiv.org/abs/2405.20799. ArXiv:2405.20799 [stat].

Ojala, Markus, Gemma C. Garriga. 2009. Permutation Tests for Studying Classifier Performance. *2009 Ninth IEEE International Conference on Data Mining*. IEEE, Miami Beach, FL, USA, 908–913. doi:10.1109/ICDM.2009.108. URL http://ieeexplore.ieee.org/document/5360332/.

Ornstein, D.S., B. Weiss. 1993. Entropy and data compression schemes. *IEEE Transactions on Information Theory* **39**(1) 78–83. doi:10.1109/18.179344. URL https://ieeexplore.ieee.org/document/179344.

Parkinson, Michael. 1980. The Extreme Value Method for Estimating the Variance of the Rate of Return. *The Journal of Business* **53**(1) 61–65. URL https://www.jstor.org/stable/2352357. Publisher: University of Chicago Press.

Pedersen, Lasse Heje. 2019. *Efficiently Inefficient: How Smart Money Invests and Market Prices Are Determined*. First paperback printing ed. Princeton University Press, Princeton, New Jersey; Oxford.

Reinsch, Christian H. 1967. Smoothing by spline functions. *Numerische Mathematik* **10**(3) 177–183. doi:10.1007/BF02162161.

Shapley, Lloyd S. 1953. A value for n-person games. H. W. Kuhn, A. W. Tucker, eds., *Contributions to the Theory of Games*, *Annals of Mathematics Studies*, vol. 28. Princeton University Press, Princeton, NJ, 307–317.

Sundararajan, Mukund, Ankur Taly, Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. doi:10.48550/arXiv.1703.01365. URL http://arxiv.org/abs/1703.01365. ArXiv:1703.01365 [cs].

Tran, Dat Thanh, Juho Kanniainen, Alexandros Iosifidis. 2022. How informative is the Order Book Beyond the Best Levels? Machine Learning Perspective. doi:10.48550/arXiv.2203.07922. URL http://arxiv.org/abs/2203.07922. ArXiv:2203.07922 [cs].

Valente, Giancarlo, Agustin Lage Castellanos, Lars Hausfeld, Federico De Martino, Elia Formisano. 2021. Cross-validation and permutations in MVPA: Validity of permutation strategies and power of cross-validation schemes. *NeuroImage* **238** 118145. doi:10.1016/j.neuroimage.2021.118145. URL https://linkinghub.elsevier.com/retrieve/pii/S1053811921004225.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. 2023. Attention Is All You Need. doi:10.48550/arXiv.1706.03762. URL http://arxiv.org/abs/1706.03762. ArXiv:1706.03762 [cs].

Appendix A

Hyperparameter Tuning

Table A.1: Hyperparameter search spaces explored via random search; all models use an Adam optimiser. $\mathcal{U}\{a, b\}$ denotes a discrete uniform draw over integers $a, \ldots, b$, and L_{seq} is the sampled sequence length. Best parameters for each model are shown in **bold**; when the choice is ambiguous (e.g., number of windows), we report it.

Model	**Hyperparameter**	**Search space**
Rough Transformer	Learning rate	{5e-4, 1e-4, **5e-5**}
	Sequence length	{**5**, 10, 20, 50}
	Number of windows	$n \sim \mathcal{U}\{3, L_{\text{seq}} - 1\}$; best **3**
	Batch size	{16, **32**, 64}
	Attention heads	{**4**, 8, 16}
	Encoder layers	{**1**, 2}
	Univariate	{**True**, False}
	Signature level	{**2**, 3}
	Embedding dimension	{4, **8**, 16}
	Warmup proportion	{**0.00**, 0.05, 0.10}
	Dropout (emb/attn/resid)	{0.1, 0.2, 0.3}; emb: **0.3**, attn: **0.3**, resid: **0.1**
Spline Transformer	Learning rate	{5e-4, 1e-4, **5e-5**}
	Sequence length	{**5**, 10, 20, 50}
	Number of windows	$n \sim \mathcal{U}\{3, L_{\text{seq}} - 1\}$; best **3**
	Batch size	{16, **32**, 64}
	Attention heads	{**4**, 8, 16}
	Encoder layers	{**1**, 2}
	Univariate	{**True**, False}

(continues on next page)

Model	Hyperparameter	Search space
	Signature level	{**2**, 3}
	Embedding dimension	{**4**, 8, 16}
	Warmup proportion	{**0.00**, 0.05, 0.10}
	Dropout (emb/attn/resid)	{0.1, 0.2, 0.3}; emb: **0.3**, attn: **0.2**, resid: **0.2**
Vanilla Transformer	Learning rate	{5e-4, **1e-4**, 5e-5}
	Sequence length	{**5**, 10, 20, 50}
	Batch size	{16, **32**, 64}
	Attention heads	{**4**, 8, 16}
	Encoder layers	{**1**, 2}
	Warmup proportion	{0.00, 0.05, **0.10**}
	Model dimension	{**128**, 256, 512}
	Feedforward dimensions	{**32**, 64, 128}
	Dropout (attn/resid/mlp)	{0.1, 0.2, 0.3}; attn: **0.3**, resid: **0.3**, mlp: **0.3**
LSTM	Learning rate	{**1e-3**, 5e-4, 1e-4}
	Sequence length	{**5**, 10, 20, 50}
	Batch size	{16, **32**, 64}
	Layers	{**1**, 2}
	Warmup proportion	{0.00, 0.05, **0.10**}
	Feedforward dimension	{128, **64**, 128}
	Hidden size	{32, **64**, 128}
	Recurrent dropout	{**0.1**, 0.2, 0.3}
Random Forest	Estimators	{50, **100**, 200}
	Max depth	{**None**, 5, 10, 20}
	Min samples per leaf	{1, 5, 10, **15**}
	Max features	{**sqrt**, log2}
	Sequence length	{**5**, 10, 20, 50}

Appendix B

Code listing

B.1 Data Formatting

```python
def get_dataset(config, seed, device, shuffle_labels_in_batch = False):
    """
    Build train/val/test loaders and metadata from the volume-sampled dataset.

    Returns
    -------
    train_loader, val_loader, test_loader : torch.utils.data.DataLoader
    seq_len : int
    num_classes : int
    num_train : int
    num_features : int
    majority_proportion, second_proportion : float
    """
    # Load volume constructred dataset and drop metadata colums. Separate features and
    ↪ labels
    df = pd.read_csv(config.vol_path)
    drop_cols = ["labels", "first_trade", "last_trade"]
    X = df.loc[:, [col for col in df.columns if col not in drop_cols]]
    y = df["labels"]

    # Split dataset into specified percentages
    n_total = len(X)
    n_train = int(n_total * (1 - config.test_size - config.val_size))
    n_val   = int(n_total * config.val_size)

    # Tensorise into (B, T, F) and align labels
    X, y = tensorise_data(X, y, config.seq_len)

    seq_len = X.shape[1]

```

```
    # Optionally add time as done in the original implementation---done only in
    ↪ univariate setting
    x = np.linspace(0, seq_len, seq_len)
    if config.add_time:
        t = (torch.linspace(0, seq_len, seq_len) / seq_len).reshape(-1, 1)
        X = torch.cat([t.repeat(X.shape[0], 1, 1), X], dim=2)

    # Signature computation. This segment is skipped for non-signature models.
    if config.use_signatures:
        if config.model == "rough_transformer_spline":
            X = ComputeSignaturesInterpolated(
                X, x, config, device,
                InterpolationOptions(
                    kind="smoothing_spline",
                    samples_per_interval=8,
                    smoothing_factor=seq_len - np.sqrt(2 * seq_len),
                ),
            )
        else:
            X = ComputeSignatures(X, x, config, device)

    num_features = X.shape[2]
    num_classes = len(np.unique(y))
    print(np.unique(y, return_counts=True)) # expose the proportions for insight

    # Optional label shuffling. Done during the permutation test to geneart the null
    ↪ distribution.
    if shuffle_labels_in_batch:
        def circular_shift(y_tensor, min_shift, rng):
            N = len(y_tensor)
            k = int(rng.integers(min_shift, N - min_shift))
            return torch.roll(y_tensor, shifts=k, dims=0)

        rng = np.random.default_rng(seed)
        y = circular_shift(y, seq_len, rng)

    # Train/val/test splits
    x_train, y_train = X[:n_train], y[:n_train]
    x_val,   y_val   = X[n_train:n_train + n_val], y[n_train:n_train + n_val]
    x_test,  y_test  = X[n_train + n_val:], y[n_train + n_val:]

    print(np.unique(y_val, return_counts=True))

    # Helper function that will help calculate the naive baselines
    def proportion(y_arr):
        classes, counts = np.unique(y_arr, return_counts=True)
        sorted_counts = np.sort(counts)
        majority_proportion = sorted_counts[-1] / counts.sum()
```

```
        second_proportion   = sorted_counts[-2] / counts.sum()
        return majority_proportion, second_proportion

    majority_proportion, second_proportion = proportion(y_train)

    # Optionally concatenate corresponding datasets for test set inference/production
    if config.train_full_model:
        x_train = torch.concat([x_train, x_val, x_test], dim=0)
        y_train = torch.concat([y_train, y_val, y_test], dim=0)
    elif not config.training:
        x_train = torch.concat([x_train, x_val], dim=0)
        y_train = torch.concat([y_train, y_val], dim=0)

    # Robust scaling
    if config.scale_data == True:
        new_x_train = x_train.reshape(-1, x_train.shape[2]).cpu().numpy()
        new_x_val   = x_val.reshape(-1,   x_val.shape[2]).cpu().numpy()
        new_x_test  = x_test.reshape(-1,  x_test.shape[2]).cpu().numpy()

        scaler = RobustScaler()
        # Fit on unique rows to avoid double counting (feaure values are never the same
    ↪ batchwise)
        scaler.fit(np.unique(new_x_train, axis=0))
        joblib.dump(scaler, f"{ASSET}_{MODEL}_scaler.pkl")

        # Transform and reshape back
        new_x_train = scaler.transform(new_x_train)
        new_x_val   = scaler.transform(new_x_val)
        new_x_test  = scaler.transform(new_x_test)

        x_train = torch.from_numpy(new_x_train).reshape(x_train.shape)
        x_val   = torch.from_numpy(new_x_val).reshape(x_val.shape)
        x_test  = torch.from_numpy(new_x_test).reshape(x_test.shape)

    # Construct the dataloaders
    train_dataset = LOBDataset(x_train, y_train)
    val_dataset   = LOBDataset(x_val, y_val)
    test_dataset  = LOBDataset(x_test, y_test)

    train_loader = DataLoader(
        train_dataset,
        batch_size=config.batch_size,
        shuffle=True,
        generator=torch.Generator().manual_seed(42),
    )
    val_loader = DataLoader(val_dataset, batch_size=config.batch_size, shuffle=False)
    test_loader = DataLoader(test_dataset, batch_size=config.batch_size, shuffle=False)

```

```
    num_train = len(x_train)

    return (
        train_loader, val_loader, test_loader,
        seq_len, num_classes, num_train, num_features,
        majority_proportion, second_proportion
    )
```

Listing B.1: Helper function that formats the data as described in the thesis prior to feeding it into the training loop.

B.2 Model Training

```
def train_one_seed(config, seed, device, shuffle_labels_in_batch = False):
    """Main training and evaluation loop for a single random seed."""
    # For reproducibility
    torch.manual_seed(seed)
    np.random.seed(seed)
    random.seed(seed)

    start_time = time.time()
    print("\n" + "=" * 50)
    print(f"Starting Training for Seed: {seed}")
    print("=" * 50)
    print("Configuration:")
    pprint.pprint(vars(config))

    # Collect the preprocessed, correctly formatted data
    (
        train_loader, val_loader, test_loader,
        seq_len, num_classes, num_samples, num_features,
        majority_proportion, second_proportion
    ) = get_dataset(config, seed, device, shuffle_labels_in_batch)

    print(
        f"\nDataset Info: Classes={num_classes}, "
        f"Training Samples={num_samples}, Features={num_features}, SeqLen={seq_len}"
    )

    keep_indices = list(range(seq_len))

    # Instantiate model, loss, optimiser, scheduler
    model = create_model(config, num_features, seq_len, num_samples, num_classes,
    ↪ device)

    criterion = nn.CrossEntropyLoss()
    optimizer = getattr(optim, config.optimizer)(
```

```
        model.parameters(),
        lr=config.lr,
        weight_decay=config.weight_decay,
    )

    steps_per_epoch = len(train_loader)
    total_steps = steps_per_epoch * config.epoch
    warmup_prop = config.warmup_proportion
    warmup_steps = max(1, int(warmup_prop * total_steps))

    def lr_lambda(step):
        # Linear warmup, then stay at 1 (i.e., constant base LR)
        return min(1.0, step / float(warmup_steps))

    scheduler = LambdaLR(optimizer, lr_lambda)

    # Early-stopping trackers
    best_val_acc = float("-inf")
    best_val_loss = float("inf")
    best_val_f1 = float("-inf")
    early_stopping_count = 0

    # Epoch schedule (adjusted if not training)
    training_epochs = (
        config.epoch
        if config.training
        else int(config.epoch * (1 - config.test_size) / (1 - config.test_size -
    ↪ config.val_size))
    )

    # Standard training loop
    for epoch in range(training_epochs):
        model.train()
        for batch in train_loader:
            inputs, labels = batch["input"].to(device), batch["label"].to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            scheduler.step()

        # Evaluation
        train_acc, train_loss, train_err, train_f1, train_wf1 = calculate_accuracy(
            config, model, train_loader, num_classes, seq_len, keep_indices, criterion,
    ↪ device
```

```
        )
        val_acc, val_loss, val_err, val_f1, val_wf1 = calculate_accuracy(
            config, model, val_loader, num_classes, seq_len, keep_indices, criterion,
    ↪ device
        )
        test_acc, test_loss, test_err, test_f1, test_wf1 = calculate_accuracy( # This
    ↪ is inspected on the final run
            config, model, test_loader, num_classes, seq_len, keep_indices, criterion,
    ↪ device
        )

        early_stopping_count += 1

        # Early-stopping decision tracking
        if val_f1 > best_val_f1:
            if config.decision == "f1":
                early_stopping_count = 0
                best_epoch = epoch
            best_val_f1 = val_f1

        if val_acc > best_val_acc:
            if config.decision == "val":
                early_stopping_count = 0
                best_epoch = epoch + 1
            best_val_acc = val_acc

        if val_loss < best_val_loss:
            if config.decision == "loss":
                early_stopping_count = 0
                best_epoch = epoch + 1
            best_val_loss = val_loss

        # --- Logging per epoch ---
        print(
            f"Epoch {epoch + 1:03}/{training_epochs} | "
            f"Train Loss: {train_loss:.4f} | Train Acc: {train_acc * 100:.2f}% | "
            f"Val Loss: {val_loss:.4f} | Val Acc: {val_acc * 100:.2f}% | Val MF1:
    ↪ {val_f1:.4f} | "
            f"Best Val Loss: {best_val_loss:.4f} | Best Val Acc: {best_val_acc *
    ↪ 100:.2f}% | Best Val F1: {best_val_f1*100:.2f}"
        )
        print(train_err, val_err, test_err)

        # Optionally record metrics for this run (used to construct loss curves)
        if config.save_metrics:
            third_proportion = 1 - majority_proportion - second_proportion
            baseline_loss = (
                -majority_proportion * np.log(majority_proportion)
```

```
                - second_proportion * np.log(second_proportion)
                - third_proportion * np.log(third_proportion)
            )
            metrics_path = f"save_metrics/{ASSET}_{MODEL}_metrics.json"

            # Load existing metrics (if any)
            if os.path.exists(metrics_path):
                with open(metrics_path, "r") as f:
                    metrics = json.load(f)
            else:
                metrics = {
                    "train_acc": [],
                    "val_acc": [],
                    "train_loss": [],
                    "val_loss": [],
                    "acc_reference": majority_proportion,
                    "loss_reference": baseline_loss,
                }

            # Append this runs results
            metrics["train_acc"].append(train_acc)
            metrics["val_acc"].append(val_acc)
            metrics["train_loss"].append(train_loss)
            metrics["val_loss"].append(val_loss)

            # Save back to disk
            with open(metrics_path, "w") as f:
                json.dump(metrics, f)

        # Early stopping
        if early_stopping_count >= config.early_stopping:
            break

    # Final logs
    print(f"\nTotal training time: {time.time() - start_time:.2f}s")
    if config.training:
        print(
            f"Stopping criteria reached. "
            f"Epoch {best_epoch} resulted in best validation accuracy of
    ↪ {best_val_acc}."
        )
    else:
        print(
            f"Stopping criteria reached. "
            f"Final Test Accuracy: {test_acc}. Final Test Loss: {test_loss}. Final Test
    ↪ F1: {test_f1}"
        )

```

```
        if config.final_model:
            with open(f"results_{ASSET}_{MODEL}.json", "w") as f:
                json.dump(
                    {"accuracy": test_acc, "loss": test_loss, "f1": test_f1},
                    f,
                )

        # Save confusion matrix for the final test model
        if config.save_confusion:
            save_confusion_matrix(
                config=config,
                model=model,
                loader=test_loader,
                num_classes=num_classes,
                seq_len=seq_len,
                keep_indices=keep_indices,
                device=device,
                prefix=f"{ASSET}_{MODEL}",
            )

        # SHAP summaries for the final model
        if config.save_shap:
            make_shap_feature_and_time_summaries(
                config=config,
                model=model,
                train_loader=train_loader,
                eval_loader=test_loader,
                keep_indices=keep_indices,
                device=device,
                out_dir="shap_plots",
                interaction_pairs=None,
            )

    if config.training:
        return best_val_acc, best_val_loss, best_val_f1, best_epoch
    else:
        return test_acc, test_loss, test_f1, best_epoch
```

Listing B.2: Training loop used to evaluate all models. Hyperparameters are set via a helper that builds a configuration from `argparse`, with defaults defined at the top of the main script.

Appendix C

Presentation

Predicting Return Direction from Volume-Sampled Limit Order Books: A Feature Engineering Approach

Artem Arshakyan

Department of Computer Science
University College London

September 22, 2025

1 / 7

What is high-frequency trading?

- High-frequency trading (HFT) lacks a single, concrete definition; it is commonly associated with ultra–low latency and very short holding horizons.
- An alternative view characterises HFT by a shift in the *sampling paradigm* to volume time rather than fixed clock time. This avoids over/under-sampling across quiet and active periods and often yields better statistical properties for bar returns (e.g., partial recovery of i.i.d. normality).
- Active investment persists despite the rise of passive funds, challenging the strong form of the efficient market hypothesis and suggesting that opportunities beyond the market portfolio remain where benefits exceed costs.
- However, any edge at these horizons is likely to come from *problem formulation* and access to *informative*, relatively under-exploited data.
- **Our approach:** engineer LOB-based features under volume time, pose a down/flat/up classification, and test whether models learn signal beyond chance while remaining interpretable.

High-Level Action Plan

- Most of our models are neural networks. Although such models are universal function approximators, this property is futile without informative features exposing the *latent manifold* our data lie on.
- Consequently, we require bespoke, informative features. Our data curation is heavily inspired by López de Prado (2018) (EWMSD labels and candidate financial features) and Gould et al. (2013).
- Specifically, we use MSFT LOBSTER data and concatenate the two standard files. We accumulate events until 10,000 shares trade, construct the features on that slice, discard it, and proceed to the next bar.
- As most models are neural networks, features are transformed via $\ln(1 + x)$ and then robust–scaled to place them near $[-1, 1]$. Exact confinement is not achieved, but training is stable.
- To assess whether models learned by chance, we run a cyclic label permutation test, control false discovery rate with the Benjamini–Hochberg procedure, and then utilise SHAP values to assess attributions when structure is present.

Main Model and Baselines

- We consider five models, most of which are variations of the Transformer architecture.
- **Main model: Rough Transformer** (Moreno-Pino et al. (2025)). It utilises rough–path signatures to replace the embedding matrix in a vanilla Transformer with compressed long- and short-term feature paths that also capture interaction effects.
- To employ rough–path signatures, the original implementation piecewise-linearly interpolates the discrete-time samples. We reproduce this and additionally extend to a *smoothing-spline* lift using the `SciPy` default smoothing factor.
- **Baselines:** the Spline Transformer, a vanilla Transformer, an LSTM, and a Random Forest. We benchmark the piecewise-linear Rough Transformer against these alternatives.

Best Model (LSTM) Results

- LSTM attains the lowest cross entropy loss and the highest macro F1; accuracy remains close to the naive baseline.
- Binary F1 is higher for the more common Class 2 vs. 0, due to recall.
- Predictions are demonstrably better than chance.

MSFT — Confusion Matrices (test)

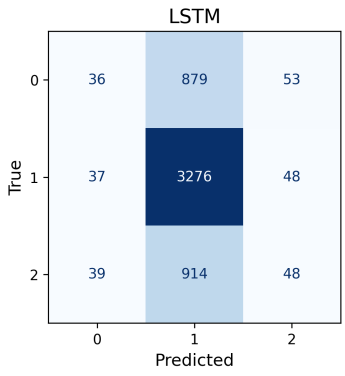


Table: LSTM permutation test (label–cyclic).

Model	***p*-value**	**Avg. *z*-score**
LSTM	0.01	−4.37
LSTM–No Execution Return	0.01	−2.38

Table: LSTM metrics (10 seeds). CE loss 95% CI in square brackets.

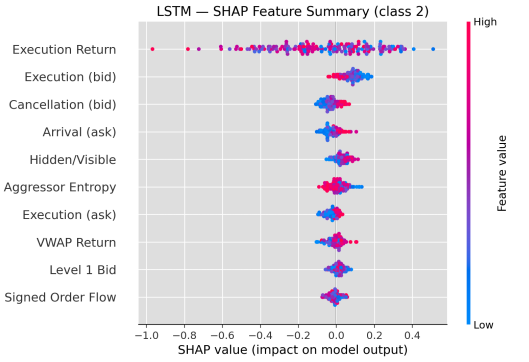
Model	**CE Loss**	**95% CI**	**Acc (%)**	**Macro F1 (%)**
LSTM	0.874 ± 0.001	[0.873, 0.874]	62.93 ± 0.00	30.15 ± 0.00

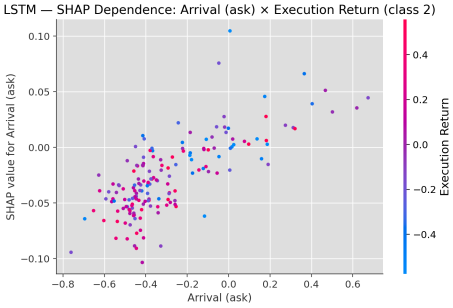
Table: LSTM per-class P/R/F1 (%, 2 d.p.).

Model	**Class 0**	**Class 1**	**Class 2**
LSTM	32.14/03.72/06.67	64.63/97.47/77.72	32.21/04.80/08.35

Interpretability and Insights

- *Execution Return* is the most influential feature by SHAP and exhibits the most nuanced interactions.
- Lower bar returns combined with higher bid arrivals push the "up" logit positively—consistent with the negative autocorrelation of returns observed at high frequency.
- We attribute the better-than-baseline cross entropy (despite muted accuracy) to the model exploiting this negative autocorrelation, generally nudging predictions in the correct direction.
- That nudge suffices to improve log-loss marginally, but not to flip the argmax away from the majority class. Eventual prediction shifts likely arise from weak, noisy signals—supported by rejection of the null even without Execution Return—and possibly from purely noisy features.





References

Gould, M. D., Porter, M. A., Williams, S., McDonald, M., Fenn, D. J., and Howison, S. D. (2013). Limit Order Books. arXiv:1012.0349 [q-fin].

López de Prado, M. (2018). *Advances in Financial Machine Learning*. John Wiley & Sons, Hoboken, NJ.

Moreno-Pino, F., Arroyo, A., Waldon, H., Dong, X., and Cartea, (2025). Rough Transformers: Lightweight and Continuous Time Series Modelling through Signature Patching. arXiv:2405.20799 [stat].