EXPLORING THE ADVANTAGES OF TRANSFORMERS FOR HIGH-FREQUENCY TRADING

A PREPRINT

Fazl Barez
Edinburgh Centre for Robotics
The University of Edinburgh
f.barez@ed.ac.uk

Paul Bilokon
Department of Computing
Imperial College London
paul.bilokon@imperial.ac.uk

Arthur Gervais
Department of Computing
Imperial College London
a.gervais@imperial.ac.uk

Nikita Lisitsyn
Department of Computing
Imperial College London
nikita.lisitsyn21@imperial.ac.uk

February 20, 2023

ABSTRACT

This paper explores the novel deep learning Transformers architectures for high-frequency Bitcoin-USDT log-return forecasting and compares them to the traditional Long Short-Term Memory models. A hybrid Transformer model, called **HFformer**, is then introduced for time series forecasting which incorporates a Transformer encoder, linear decoder, spiking activations, and quantile loss function, and does not use position encoding. Furthermore, possible high-frequency trading strategies for use with the HFformer model are discussed, including trade sizing, trading signal aggregation, and minimal trading threshold. Ultimately, the performance of the HFformer and Long Short-Term Memory models are assessed and results indicate that the HFformer achieves a higher cumulative PnL than the LSTM when trading with multiple signals during backtesting[1].

1 Introduction

Forecasting Financial Time Series (FTS) has been of interest to financial market participants who are interested in making profitable trades on the financial markets. It has historically been approached using stochastic and machine learning models. Stochastic methods include linear models such as Autoregressive Integrated Moving Average (ARIMA) [1] that support non-stationary time series and non-linear models, including the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) [2] model. Machine learning methods are data-driven approaches, among which Recurrent Neural Networks (RNNs) [3], more specifically, Long Short-Term Memory (LSTM) networks [4], have been especially popular for time series prediction. Periodically, new deep learning models are being adopted in quantitative research to find the most accurate models in FTS forecasting that would lead to more efficient trading strategies.

Recently, a new type of deep learning [5] architecture called Transformer [6], relying on Attention [7], was introduced for Natural Language Processing (NLP) applications. Transformers have since been used in other applications such as computer vision tasks [8] and more recently in time series forecasting. This paper will focus on the application of Transformers in high-frequency FTS forecasting.

FTS are characterized by properties including frequency, auto-correlation, heteroskedasticity, drift, and seasonality [9]. The frequency of trading strategies ranges from milliseconds to minutes and days. Most studies on FTS forecasting [10, 11] focus on minute frequencies as millisecond trading data is expensive. For

[1]The code will be publicly available upon publication

the results of this paper to be in line with real-world trading conditions, we will use millisecond frequency tick price and level 2 Limit Order Book (LOB) data [12] which will allow us to focus on the bid and ask prices and quantities of the first ten levels of the LOB. We will be using data collected from the Binance cryptocurrency exchange for the Bitcoin - USDT trading pair.

Traders require automated systems to make decisions based on the order book and tick price data at millisecond frequencies. Stochastic modeling has long been used to describe prices as exogenous random variables, however, it relies on assumptions that may oversimplify the complexity of financial markets. Contrary to stochastic modeling, deep learning forecasting relies on a data-driven approach to learn past dynamics to predict future behavior. Deep learning approaches have shown promising results in High-Frequency Trading (HFT) strategies [10]. Currently, LSTMs are the most widely used deep learning architecture for FTS forecasting [13]. However, LSTMs are affected by vanishing and exploding gradient problems for long time series [14], and they are sequential structures that are difficult to parallelize.

Transformers use multi-head Attention to capture long-term dependencies and are naturally parallelizable, which makes them a good candidate for low-latency trading strategies. However, Transformers were not specifically engineered for modeling local sequential structures and require finely tuned position embeddings. Modeling short- and long-range temporal dependencies while accounting for FTS properties (e.g., seasonality and auto-correlation) remains an open question [15, 16].

In this paper, we compare the performance of existing deep learning architectures for FTS forecasting. We then introduce an improved deep learning model called the HFformer, which combines certain features of previously compared architectures. Finally, we compare the performance of the HFformer to existing architectures in the context of FTS forecasting. We focus on describing the collected cryptocurrency data and detailing the pre-processing of the data before feeding it into deep learning models. Then, we cover the LSTM and original Transformer encoder-decoder architecture and compare their forecasting performance. We focus on understanding more recent Transformer architectures precisely engineered for time series forecasting: Autoformer and FEDformer. We evaluate the idiosyncrasies of these novel architectures and compare their performance to the original Transformer. We combine some novelties introduced by time series Transformer architectures to create a Transformer specifically optimized for HFT, which we refer to as HFformer. Finally, we compare the HFformer's performance to the most currently used deep learning architecture for FTS forecasting, the LSTM, on larger datasets. We also compare the trades done by both architectures. We conclude by presenting possible trading strategies and assessing their performance.

2 Background and Related Work

2.1 FTS

The world financial markets have undergone a technological revolution in the past decades. The execution time of market orders decreased from 25 ms in 2000 to less than 0.017 ms in 2017 [17]. The trades are spaced randomly and endogenously [18] as more trading occurs during market openings and closings, breaking news, and abrupt price changes. This leads to auto-correlated prices and quantities. FTS are non-stationary as they exhibit seasonality at different scales linked to the market opening hours, underlying assets, and economic events. Properties such as trend, drift, auto-correlation, and heteroskedasticity make FTS forecasting a challenging problem.

2.2 Traditional Methods for FTS

Stochastic FTS forecasting relies on linear and non-linear methods. Linear models include Moving Average (MA), Autoregressive (AR), and ARIMA [19] models. Linear models can solve stationary and non-stationary time series. Non-linear models such as GARCH and GARCH-based models are used to forecast stock market volatility exhibiting heteroskedasticity [20].

Currently, LSTMs are successfully used for minute-frequency FTS forecasting [21] for one- and multi-step forecasting [22]. Convolutional Neural Networks (CNNs) and LSTM-CNNs are used to extract information from LOBs [10]. Spiking Neural Networks (SNNs) predict price spikes for HFT strategies [23].

Attention layers have been combined with LSTMs [24] for NLP sentiment classification tasks. Recently, LSTM encoder-decoder architectures with dual Attention have been used to make the encoder and decoder hidden state selection more relevant for FTS forecasting [11].

2.3 Transformers for FTS

Initially introduced for NLP, the Transformer has shown state-of-the-art performance for forecasting time series [25]. The Transformer requires a positional encoding, such as Time2Vec [26], to be added to the

input embeddings to model time series. Novel Transformer designs [27] have focused on reducing the $O(L^2)$ complexity by using causal convolutions or sparse bias in the Attention module to process longer sequences.

Other structural changes were made to improve time series forecasting. The Autoformer [15] has a modified auto-correlation Attention module that selects the top-k similar sub-sequences. The Temporal Fusion Transformer (TFT) [28] uses static covariate encoders, gating layers for noise suppression, and recurrent layers for local processing. The FEDformer [16] performs a Fourier and Wavelet time series decomposition using the Attention mechanism.

3 Exploratory Data Analysis

3.1 FTS data

Level 2 LOB data is collected from the Binance cryptocurrency exchange for the BTC-USDT trading pair using Binance's 100 ms LOB web socket[2].

Most of the collected LOB snapshots for the BTC-USDT trading pair are within 100 ms. We then proceed to compute the weighted midprice defined as follows:

$$Midprice_t = \frac{q_t^{a;1} \cdot p_t^{a;1} + q_t^{b;1} \cdot p_t^{b;1}}{2} \tag{1}$$

where $p_t^{a;1}$, $q_t^{a;1}$, $p_t^{b;1}$, and $q_t^{b;1}$ are the bid price and quantity and ask price and quantity of level 1 of the LOB at time t.

We process the collected LOB data by dropping snapshots with the same consecutive midprice value as the analysis will be done at trade time rather than wall clock time. This is done because we are interested in predicting the values at which the future trades will occur, and this brings the distribution of the observed values closer to a normal distribution (cf. subsubsection 3.1.1). In this paper, the terms "midprice" and "weighted midprice" will be used interchangeably.

3.1.1 Log-Returns

One challenge with FTS is that the range of values varies significantly, even for the same financial asset. If we want to apply a deep learning method to forecast FTS, we need to transform the time series to remove the non-stationarity. Usually, three price time series transformations are proposed:

- price differencing: $d_{t+1} = p_{t+1} - p_t$
- returns: $r^*_{t+1} = \frac{p_{t+1} - p_t}{p_t}$
- log-returns: $r_{t+1} = \log(p_{t+1}) - log(p_t)$

All three of these methods remove the non-stationarity of prices. However, using price differencing preserves the units and will be proportional to the underlying price of the asset, which is undesirable in the case of Bitcoin because of its high volatility. On the contrary, returns and log-returns are unitless as it is a ratio of two prices. Returns represent the change in asset price as a percentage and are therefore widely used in finance for their convenience.

From now on, we focus on forecasting log-returns over different time horizons. We define log-returns the following way:

$$r_{t+\tau} = \log(p_{t+\tau}) - \log(p_t) = \log\left(\frac{p_{t+\tau}}{p_t}\right) \tag{2}$$

where $\tau \in \{1, ..., 30\}$. τ is the value of the forecasting horizon.

The range of considered forecasting horizons can be justified using the Augmented Dickey-Fuller (ADF) stationarity test, which tests for different horizons of the log-returns. As the prediction horizon increases, the information contained in the observed prices, quantities, and historical returns becomes less informative for future predictions (cf. Figure 1) because the time series becomes less stationary. A lower value means that the time series exhibits stronger stationary properties.

[2]More details on Binance's LOB web socket can be found at https://www.binance.com/en/binance-api.

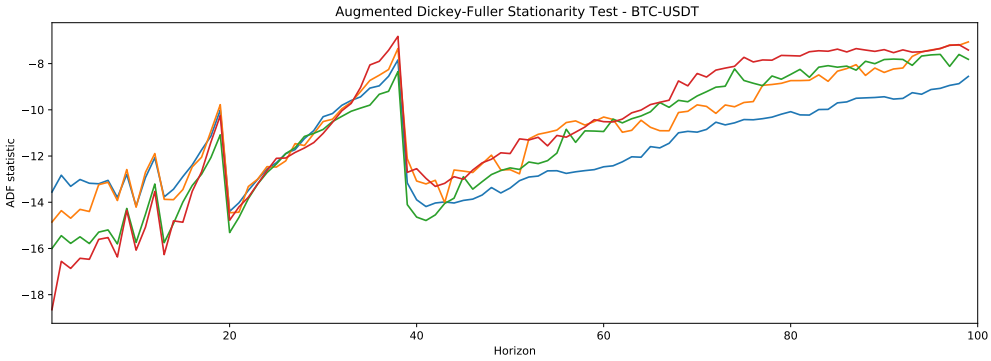

Figure 1: Augmented Dickey-Fuller stationarity test for different log-return horizons of four different samples of 9000 LOB snapshots.

3.1.2 LOB

The LOB contains information about the bid and ask prices and the quantities offered at those prices (cf. Figure 2). Market participants can choose to place different types of orders. Market orders are executed at the current market price if the offered quantity allows; otherwise, the order will be partially filled until completion at higher levels of the LOB for buy orders and lower levels of the LOB for sell orders. This can sometimes lead to price slippage, resulting in a less desirable average fill price. Another strategy is to place a limit order which will be executed only at the specified price or better as mentioned in the order. Depending on the limit price, these orders will appear at a certain level in the LOB.

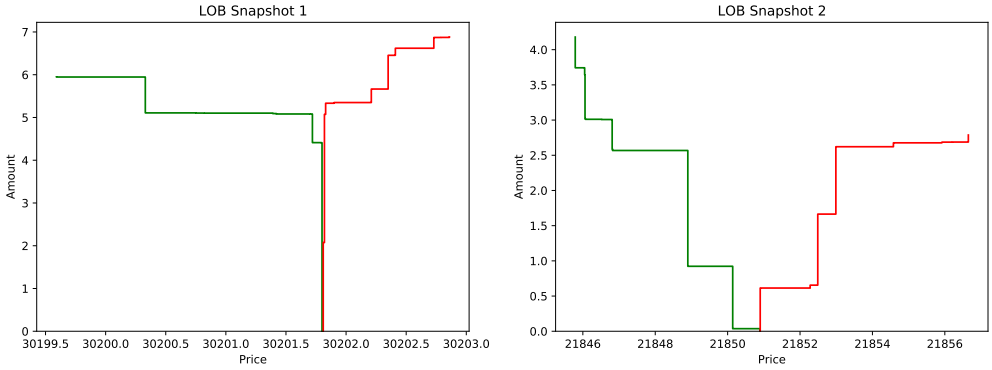

Figure 2: Bid-ask pressure for 10 levels of bid and ask for BTC-USDT.

Figure 2 shows two examples of the bid-ask pressure. In the first case, the midprice will have a greater resistance to change than in the second case since a larger amount of BTC would need to be bought or sold to move the midprice.

From Figure 3, we notice that as the bid pressure builds up in the LOB, the weighted midprice starts to move higher and vice-versa for the ask pressure. This leads us to explore the predictive power of the information contained in the LOB for forecasting future log-returns.

3.1.3 Normalization of FTS

To improve the performance of machine learning methods, input data needs to be normalized. Traditionally, the normalization parameters from the training set are applied to the test set (e.g., the mean and standard deviation of the training data). However, as in the case of the BTC-USDT pair, the mean and standard deviation of the prices and quantities fluctuate significantly. Taking this into account, we propose to use online normalization.

The online normalization of the input data is done by computing the mean and standard deviation of the data contained in the look-back window. We then normalize the data in the look-back window using the computed mean and standard deviation. Later, we will test the performance of the forecasting models for look-back windows of sizes 100, 200, 300, and 400 timesteps (cf. Figure 12). During data normalization, it is important not to compute any estimates based on future values that we want to forecast to avoid contaminating the training set.

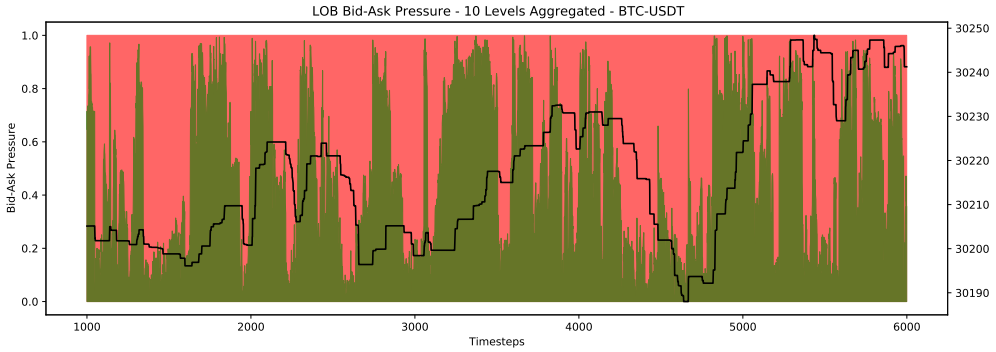

Figure 3: LOB bid-ask pressure for 10 aggregated levels.

4 LSTM and Transformers for FTS

4.1 LSTM

Recurrent Neural Networks (RNNs) are neural networks that were developed to work with sequential data to model dependencies of arbitrary length. The goal is to learn the mapping between $x_{1:T}$ and $y_{1:T}$ by encoding the input sequence into a hidden state h_t that is then decoded into y_t:

$$\boldsymbol{h}_t = \phi_h(W_h \boldsymbol{h}_{t-1} + W_x \boldsymbol{x_t} + \boldsymbol{b}_h) \tag{3}$$

$$\boldsymbol{y}_t = \phi_y(W_y \boldsymbol{h}_t + \boldsymbol{b}_y) \tag{4}$$

where $W_h, W_x, W_y, \boldsymbol{b}_h, \boldsymbol{b}_y$ are learned parameters and ϕ_h and ϕ_y are activation functions.

However, RNNs suffer from vanishing/exploding gradients due to the backpropagation mechanism used during Gradient Descent (GD). GD is an iterative optimization algorithm used to find a given function's local optimum to reduce the difference between the predicted and true values. Various loss functions are used to compute the difference between the true and predicted values, which is then propagated back to modify the weights of the model. This issue is mitigated by using memory cell states and gating mechanisms in LSTMs:

$$\boldsymbol{f}_t = \sigma(W_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x_t}] + \boldsymbol{b}_f) \tag{5}$$

$$\boldsymbol{i}_t = \sigma(W_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x_t}] + \boldsymbol{b}_i) \tag{6}$$

$$\boldsymbol{o}_t = \sigma(W_o \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x_t}] + \boldsymbol{b}_o) \tag{7}$$

$$\tilde{\boldsymbol{c}}_t = \tanh(W_c \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x_t}] + \boldsymbol{b}_c) \tag{8}$$

$$\boldsymbol{c}_t = \boldsymbol{f} \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{9}$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} \odot \boldsymbol{c_t} \tag{10}$$

where $W_f, W_i, W_o, W_c, b_f, b_i, b_o, b_c$ and $\sigma\cdot$ is the sigmoid activation function.

4.2 Transformer

4.2.1 Attention

A scaled dot-product Attention [6] mechanism is a key-value lookup based on a query matrix $\boldsymbol{Q}$, key matrix $\boldsymbol{K}$, and value matrix $\boldsymbol{V}$ described as:

$$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}; \alpha) = \alpha\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_q}}\right)\boldsymbol{V} \tag{11}$$

where $\boldsymbol{Q} \in \mathbb{R}^{N \times d_q}, \boldsymbol{K} \in \mathbb{R}^{M \times d_q}, \boldsymbol{V} \in \mathbb{R}^{M \times d_v}$, and $\alpha(.)$ is an activation function applied row-wise. d_q and d_v represent the dimensions of the query and value vectors respectively. The $softmax(.)$ activation function is used in Transformers and is defined as:

$$softmax(x_i) = \frac{\exp(xi)}{\sum_j \exp(x_j)} \tag{12}$$

Intuitively $softmax(.)$ transforms a vector $\boldsymbol{x}$ into a probability vector for which the sum of elements adds up to 1.

Transformers, compared to RNNs, are easily parallelizable and can be fed multiple consecutive time series in a batch to encode and decode. However, providing consecutive time series may prevent the Transformer from learning, as it could use the time series in the batch to see the "future." Masked Attention prevents the Transformer from seeing future values:

$$MaskedAttention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}; \alpha, \boldsymbol{M}) = \alpha\left(mask\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_q}}, \boldsymbol{M}\right)\right)\boldsymbol{V} \tag{13}$$

where $\boldsymbol{M}_{nm}$ is 0 for masking and 1 otherwise. When $\boldsymbol{M}_{nm} = 0$, set $(\boldsymbol{Q}\boldsymbol{K}^T)_{nm} = -\infty$ which in the case of $softmax(.)$ will result in $\exp\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_q}}\right) = 0$. Therefore, the value $\boldsymbol{v}_m$ will not contribute to the Attention output for query $\boldsymbol{q}_n$.

Transformers combine multiple Attention heads into a multi-head Attention module where each head captures different parts of the sequence, which allows for coarser and finer encodings of the time series.

4.2.2 Position Encoding

Notice that scaled dot-product Attention is permutation equivariant:

$$Attention(\boldsymbol{P}\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}; \alpha) = \alpha\left(\frac{\boldsymbol{P}\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_q}}\right)\boldsymbol{V} = \boldsymbol{P}\alpha\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_q}}\right)\boldsymbol{V} = \boldsymbol{P}Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}; \alpha) \tag{14}$$

As the ordering information matters in time series forecasting, where we input a sequence of queries $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_N$, we construct a modified version of the query matrix $\boldsymbol{Q}$:

$$\widetilde{\boldsymbol{Q}} = (\widetilde{\boldsymbol{q}}_1, \ldots, \widetilde{\boldsymbol{q}}_N)^T, \quad \widetilde{\boldsymbol{q}}_n = f(\boldsymbol{q}_n, PE(n)) \tag{15}$$

where $f(.)$ is a summation and concatenation, and $PE(n)$ is the position encoding of input with index n, which could be learned or computed. We use sinusoid embedding to encode FTS ordering information.

4.2.3 Layer Normalization

Layer normalization [29] is used to stabilize the training of the neural network.

Let $\boldsymbol{y} = \{y_1, \ldots, y_n\} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ be the output of a feed-forward layer before going through an activation function. The layer normalization is defined as:

$$y \leftarrow \frac{y - \hat{y}}{\sigma}, \quad \hat{y} = \frac{1}{n}\sum_{k=1}^{n} y_k, \quad \sigma = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(y_k - \hat{y})^2} \tag{16}$$

In Transformers, layer normalization is applied with a residual connection:

$$Add\&Norm(x) = LayerNorm(x + Sublayer(x)) \tag{17}$$

where $Sublayer(.)$ can either be a multi-head Attention layer or a point-wise feed-forward network.

4.2.4 Point-wise Feed-forward Network

Transformers use the point-wise feed-forward network as the feed-forward layer. The output of the multi-head Attention layer after the Add&Norm layer is a $N \times d_{out}$ matrix which contains the Attention results for N queries. The point-wise feed-forward network processes this matrix row by row.

4.3 Novel Transformer Architectures

4.3.1 Autoformer

The Autoformer uses a new type of Attention based on auto-correlation in its encoder and decoder. The encoder eliminates the long-term trend by series decomposition using the Fast Fourier Transform and focuses on modeling seasonal patterns. The past seasonal and trend information from the encoder is then used by the decoder to predict future values.

Compared to the Transformer's full Attention module (cf. Figure 4), the auto-correlation Attention (cf. Figure 5) achieves a reduced complexity of $O(L\log(L))$ compared to $O(L^2)$ for a length-L time series. The performance improvement is due to querying only the top $\log(L)$ entries based on the series periodicity.

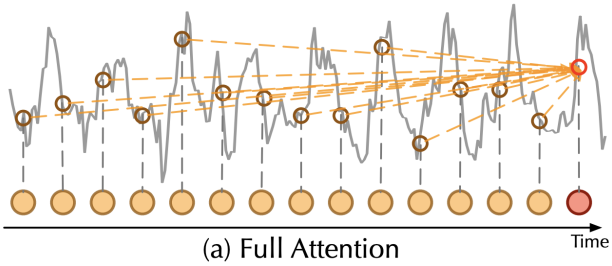

Figure 4: Attention module in a Transformer [15].

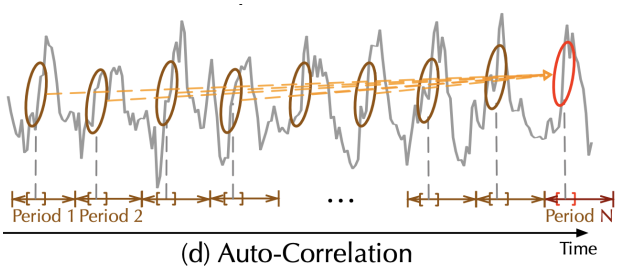

Figure 5: Auto-correlation Attention module in Autoformer [15].

4.3.2 FEDformer

The FEDformer, similar to the Autoformer, uses seasonal-trend decomposition to predict future values. The FEDformer uses either the Fast Fourier Transform or the Wavelet Transform to map the time series into the frequency domain. It then randomly queries the frequency domain Attention module for high-frequency and low-frequency components, which reduces the complexity from $O(L^2)$ to $O(L)$ for length-L time series. The goal behind using the frequency domain is to achieve an optimal mix between high-frequency components, which are more affected by local noise, and low-frequency components, which are more affected by the overall trend of the time series.

4.4 HFformer

Having evaluated existing Transformer-like architectures, we combine some of their different architectural components into a new Transformer-like architecture called the HFformer. Below is an illustration of the architecture of the HFformer.

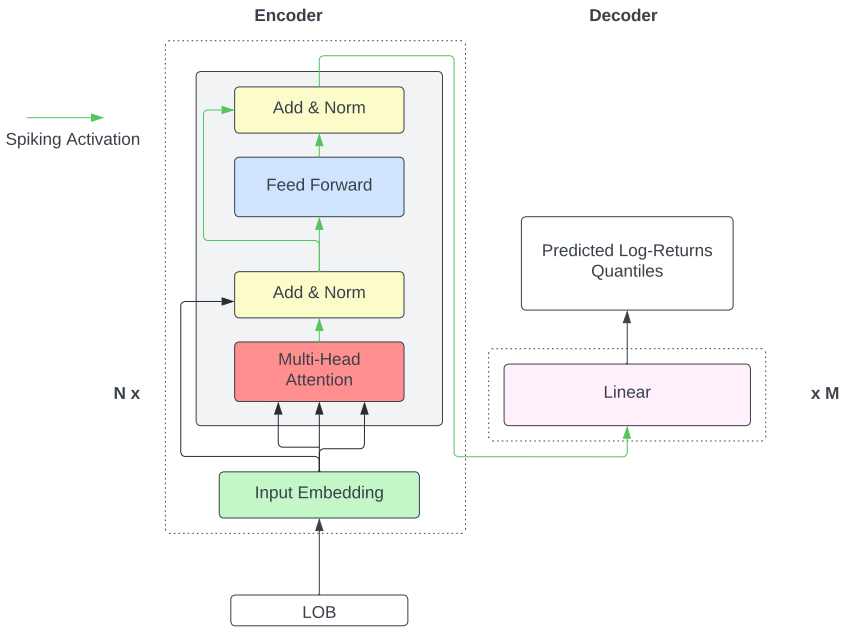

Figure 6: HFformer - Architecture.

4.4.1 Transformer Encoder with Spiking Activation

The HFformer's encoder is based on a Transformer encoder with activation functions that were modified to be spiking:

$$\mathit{SpikingActivation}(x) = \begin{cases} x, & \text{if } x \geq \mathit{threshold}, \\ 0, & \text{if } x < \mathit{threshold}. \end{cases} \tag{18}$$

where the $threshold$ is learned during training.

We use a combination of the spiking activation with a PReLU activation function. The Pytorch-Spiking Python library[3] provides the implementation of the spiking activation. Due to the noisy nature of the LOB data, a spiking activation function reduces the noise propagated through the Transformer encoder by learning a passing threshold below which the information is not propagated in the encoder.

4.4.2 Linear Decoder

The linear decoder contains two linear layers connected by a PReLU activation function. The decoder needs to output a single value for each forecast horizon as we forecast log-returns. Shifting from an autoregressive decoder to a linear decoder helps to reduce error propagation and decrease the training time.

4.4.3 No Positional Encoding

The positional encoding was removed from the model. As a result, the model's performance was improved, and the training time was reduced as the number of trainable parameters decreased. Positional encoding was introduced for NLP tasks. It is used as an augmentation technique for the self-Attention mechanism, which is invariant to the sequence order. However, as we no longer require an autoregressive decoder since we use a linear decoder, the positional encoding becomes less relevant. Additionally, the amount of information provided to the Transformer encoder increases with the position dimension, which could serve as a form of positional encoding [30].

4.4.4 Quantile Loss

The quantile loss is defined as follows:

$$L(\hat{y}_i^p, y_i) = max[q(\hat{y}_i^p - y_i), (q-1)(\hat{y}_i^p - y_i)] \tag{19}$$

where $\hat{y}_i^p$ is the model's output for a given quantile p and y_i is the true value. The mean reduction is used for the loss.

The HFformer works with MSE and MAE loss. However, quantile loss allows forecasting an interval around the predicted value. This is helpful when the inter-quartile range becomes larger than the mean inter-quartile range, which could signify a low certainty about the forecasted value and therefore one might decide not to enter the trade.

5 Experiments

5.1 Data

To compare the performance of different deep learning architectures, we use a subset of 100,000 LOB snapshots, where 80,000 LOB snapshots are used for training and 20,000 for validation (cf. Figure 7 and Figure 8).

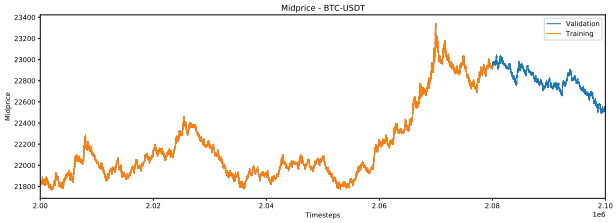

Figure 7: Subset of LOB snapshots used to assess model performance - Midprice.

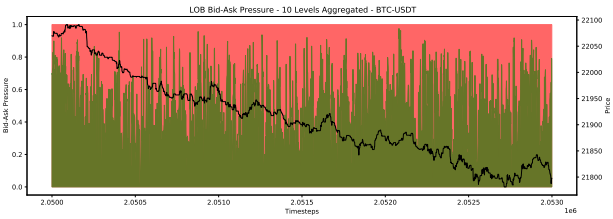

Figure 8: Subset of LOB snapshots used to assess model performance - Bid-Ask pressure.

The same subset of LOB snapshots is used later for comparing different Transformer architectures.

5.2 Setup

The input data fed into the models consists of 38 features:

- 9 bid prices and the 9 associated quantities
- 9 ask prices and the 9 associated quantities

[3]More details about the Pytorch-Spiking Python library can be found at https://github.com/nengo/pytorch-spiking.

- historical lagged log-return variable $\log\left(\frac{Midprice_t}{Midprice_{t-\tau}}\right)$ where τ is the forecast horizon
- the weighted midprice

The model's output is the log-return for a given forecast horizon $\tau \in \{1, ..., 30\}$.

The training uses the adaptive moment estimation Adam with Weight Decay (AdamW) [31] optimizer. The AdamW optimizer estimates the first and second moment of the gradient and adapts the learning rate for each input feature. The decoupled weight decay improves the convergence to an optimum.

The Mean Squared Error (MSE) and Mean Average Error (MAE) loss functions were considered for the LSTM, Transformer, Autoformer, FEDformer, and HFformer models. Models trained with the MSE loss, although more sensitive to outliers than the MAE loss, yielded a higher out-of-sample R^2 score.

5.3 Evaluation

Out-of-sample R^2 is used to assess the performance of the models for forecasting log-returns at different horizons. R^2 has the advantage of being interpretable relative to a baseline score. However, R^2 is sensitive to outliers. Finally, after each training epoch, the model's performance is evaluated using training and validation sets to avoid overfitting.

5.4 Results

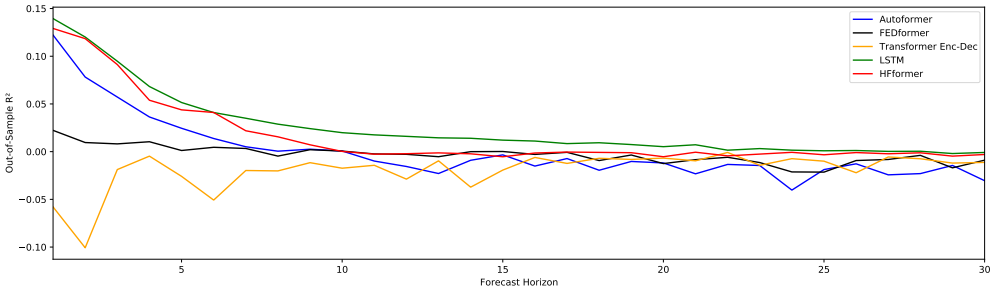

Figure 9: Autoformer vs. FEDformer vs. Transformer vs. LSTM vs. HFformer - Log-returns.

From Figure 9, we notice that the LSTM outperforms the Transformer model for all forecasting horizons. This could be due to the high noise-to-signal ratio that the LOB contains. The Transformer is more sensitive to noise as it is a more complex neural network which may lead it to overfit the training data even if the training is done using a validation set to reduce the chances of overfitting. We will later see that the Transformer's encoder performance improves with noise-reducing activation functions and larger training sets.

Among Transformer-like architectures, the Autoformer achieves the best performance on shorter forecasting horizons (cf. Figure 9). The Autoformer and FEDformer require an additional input feature which is the date and time timestamp, as both of these models use a time embedding. The frequency of the target time series (e.g., seconds, minutes, and hours) is used as a hyperparameter.

The HFformer outperforms the other Transformer-like architectures, however, it underperforms the LSTM (cf. Figure 9), however, as the size of the training dataset increases, the HFformer attains better forecasting results than the LSTM during backtesting.

The hyperparameters are listed in Table 5.

5.5 Ablation Study

We conduct an ablation study to determine the impact of different suggested improvements used in the HFformer compared to the original Transformer architecture.

The following ablations are made:

- the spiking activation combined with the PReLU activation function is replaced with a PReLU activation function
- the positional encoding is added back
- the linear decoder is replaced with the autoregressive Transformer decoder

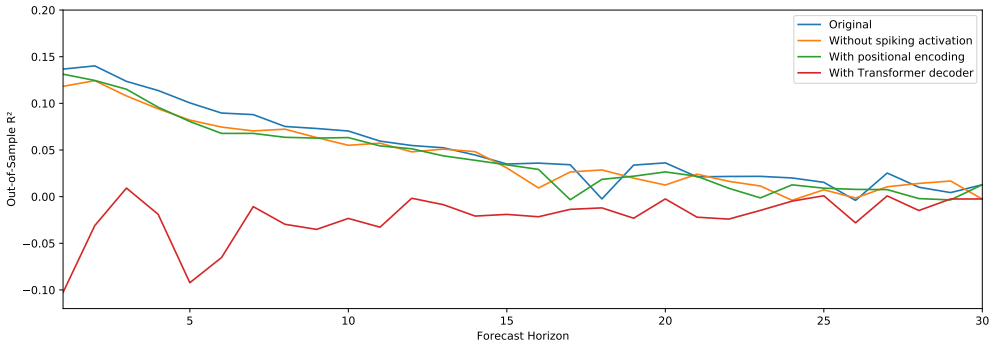

Figure 10: Ablation study for the HFformer - Log-returns.

We notice from Figure 10 that a major improvement in the R^2 score is achieved by replacing the Transformer decoder with a linear decoder. Adding a spiking activation and removing the positional encoding result in marginal improvements.

5.6 Look-back Window: LSTM vs. HFformer

We experiment with the size of the look-back window to find out which size will yield the best performance for longer forecasting horizons. The LSTM and HFformer models are trained and tested with different look-back window sizes using the same data as previously.

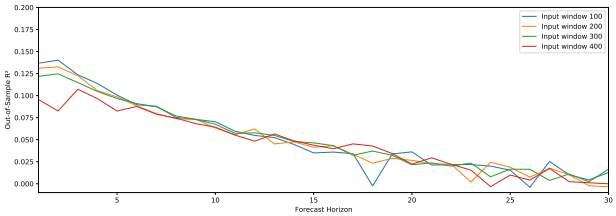

Figure 11: Look-back window size impact on the HFformer - Log returns.

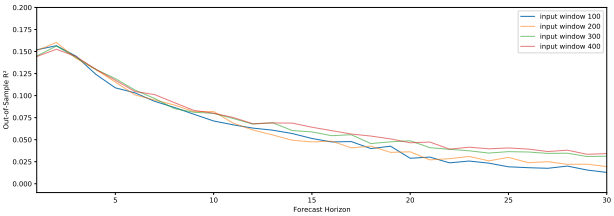

Figure 12: Look-back window size impact on the LSTM - Log-returns.

We notice that increasing the size of the look-back window positively affected the performance of the LSTM (cf. Figure 12). However, there was no noticeable change for the HFformer (cf. Figure 11).

5.7 Training and Validation Sets

The LSTM and HFformer models are trained on sets of the following size:

- Training set with 80,000 LOB snapshots and validation set with 20,000 LOB snapshots
- Training set with 300,000 LOB snapshots and validation set with 60,000 LOB snapshots
- Training set with 600,000 LOB snapshots and validation set with 120,000 LOB snapshots

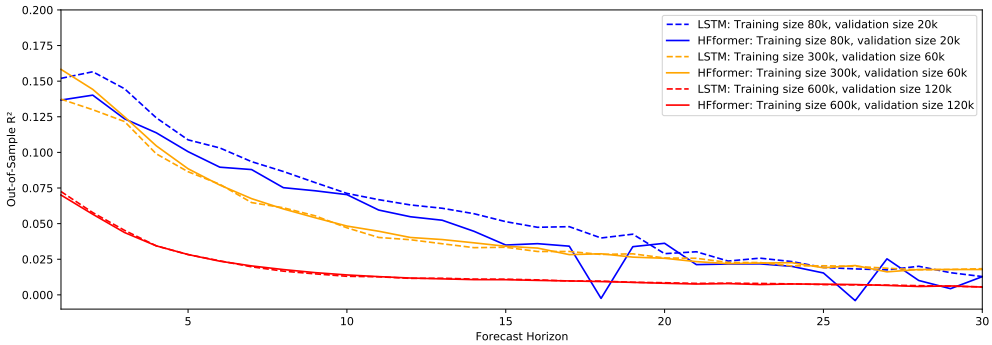

Figure 13: Size of validation and test sets impact on the HFformer and LSTM - Log-returns.

From Figure 13, we notice that as the training and validation sets increases the absolute performance of both models decreases. However, it is important to consider larger training and validation sets, as during live trading, it can be technically challenging to continuously retrain the model and may lead to overfitting. The LSTM outperforms the HFformer on smaller training and validation datasets. The HFformer's forecasting performance improves relative to the LSTM's as the size of the training and validation sets increases.

5.8 Classification Performance

Previous comparisons have been based on R^2, which assesses the regressional quality of the model. We now proceed to study the classification performance of the HFformer and LSTM. For the following comparisons, we use the HFformer and LSTM models trained on 600,000 LOB snapshots and validated on 120,000 LOB snapshots. The classification performance is assessed on a test set of size 200,000 LOB snapshots.

We define two classes:

- buy, when the predicted log-return is positive
- sell, when the predicted log-return is negative

The ratios are the true positive rates for each class.

The first comparison is based on true positives and false positives. The second comparison is a mix of classification and regression. The computed ratios are weighted ratios where the weights are the true values of the log-returns. The motivation behind this comparison is to assess whether both of the models can classify relatively large log-returns correctly. This is important in the case of volatile assets such as Bitcoin since, if the model fails to predict large log-return changes, the trading strategy will result in large drawdowns and limited upsides.

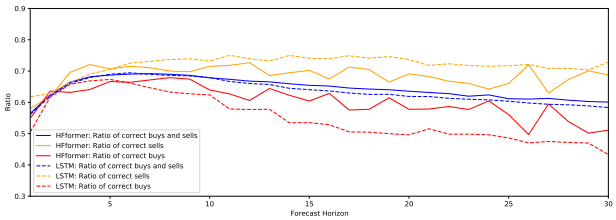

Figure 14: Classification performance of the HFformer and LSTM - Log-returns.

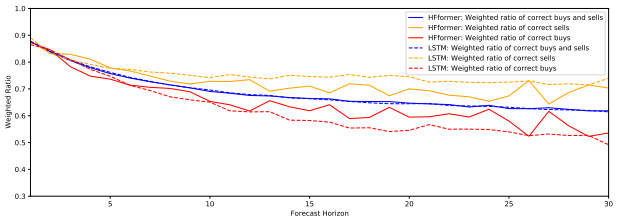

Figure 15: Classification performance with weights of the HFformer and LSTM - Log-returns.

We notice that, on average, the HFformer exhibits a slightly better classification performance than the LSTM (cf. Figure 14). For the weighted ratio, the overall performance of the HFformer is similar to the LSTM(cf. Figure 15). However, the HFformer has a smaller gap between the correct buys and correct sells ratios. We will later examine the impact of having a model that flags buy and sell opportunities in a more balanced way.

5.9 Backtesting

5.9.1 Strategies

The HFformer and LSTM models were trained on June 2022 data and are now backtested on BTC-USDT LOB snapshots collected over 2 days (July 21st and 22nd, 2022).

The backtesting is based on the following assumptions:

- No trading fee, which is a realistic assumption as there are no spot trading fees on Binance for the BTC-USDT pair
- The delay from receiving the snapshots, processing them, and placing a trade is around 200 ms, which is approximately equivalent to 2 ticks for the LOB snapshots that were collected
- The orders are market orders
- 0.0002% of the amount bought then sold for long trades (and vice-versa for short trades) is subtracted from the Profit and Loss (PnL) to simulate price slippage
- A fixed quantity of 0.1 BTC is traded

The following trading strategies are considered:

- Strategy 1: the trading horizon is 28 ticks. We use the HFformer and LSTM models trained on the specific horizon to generate a trade signal. If the signal is positive, we open a long position of 0.1 BTC with a delay of 2 ticks and close it after 28 ticks plus 2 ticks of delay. Similarly, if the signal is negative, we open a short position and proceed with a similar method to the long strategy. While running strategy 1, when we create a scatter plot of profitable and unprofitable trades based on start and end signals, we notice a relationship between the sign of the signals and the profitability of trades (cf. Figure 17).
- Strategy 2: the trading horizon is 28 ticks. We use three HFformer and three LSTM models trained on the specific horizon and +/- 2 horizons. Only if all 3 signals are positive, we open a long position of 0.1 BTC with a delay of 2 ticks and close it after 28 ticks plus 2 ticks of delay. Similarly, if all 3 signals are negative, we open a short position and proceed with a similar method to the long strategy.
- Strategy 3: similar to Strategy 2, however, instead of using 3 signals to trade, we use 5 signals: signals for horizons 26, 27, 28, 29, and 30.

5.9.2 Results

Strategy 1 results in a final PnL of -10.42 USDT and a total of 13,743 trades for the HFformer and in a final PnL of -32.54 USDT and a total of 13,743 trades for the LSTM. From Figure 16 and Figure 17, we notice that most profitable trades happen when the HFformer model outputs start and end signals of the same sign. One possible explanation is that profitable trades occur when the upward or downward trend predicted by the model lasts longer than the forecasting horizon. Therefore, one possible improvement to this trading strategy is to use multiple forecasting horizons to assess whether to go into a trade. The multiple forecasting horizons can include short forecasting horizons (e.g., 26 and 27), which are shorter than the main forecasting horizon (e.g., 28 in strategy 1) to reduce the chance of falsely flagging a trading opportunity. Additionally, longer forecasting horizons (e.g., 29 and 30) can be used to confirm that the upward or downward trend duration will last beyond the trade delay (e.g., 2 ticks in strategy 1). This leads us to strategy 2.

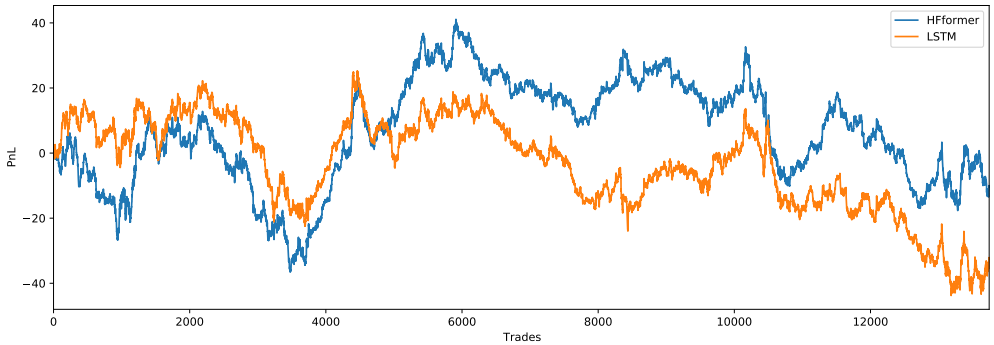

Figure 16: Cumulative PnL - Strategy 1 - HFformer vs. LSTM.

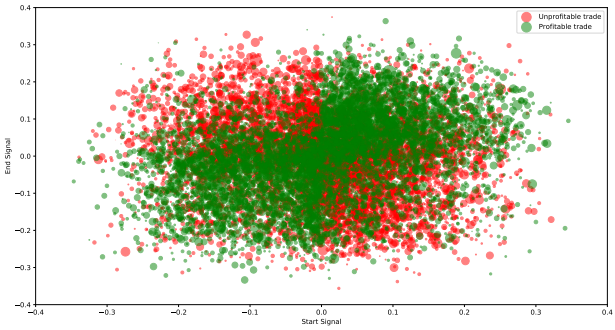

Figure 17: Relationship between PnL and start and end signals outputted by the model - Strategy 1 - HFformer.

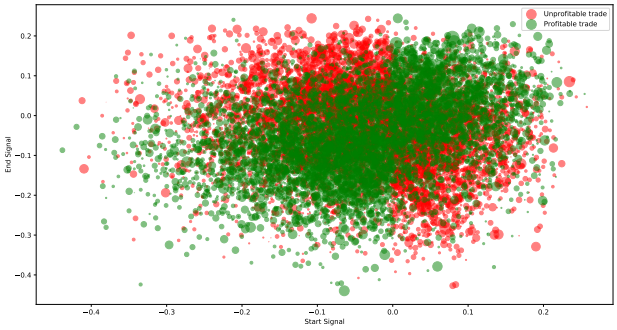

Figure 18: Relationship between PnL and start and end signals outputted by the model - Strategy 1 - LSTM.

Strategy 2 results in a final PnL of 120.04 USDT and a total of 11,928 trades for the HFformer and in a final PnL of 9.67 USDT and a total of 12,932 trades for the LSTM. We notice an improvement in the PnL and a reduction in the total number of trades. The total number of trades decreases as all three models with different forecast horizons need to output a signal of the same sign for a trade to occur.

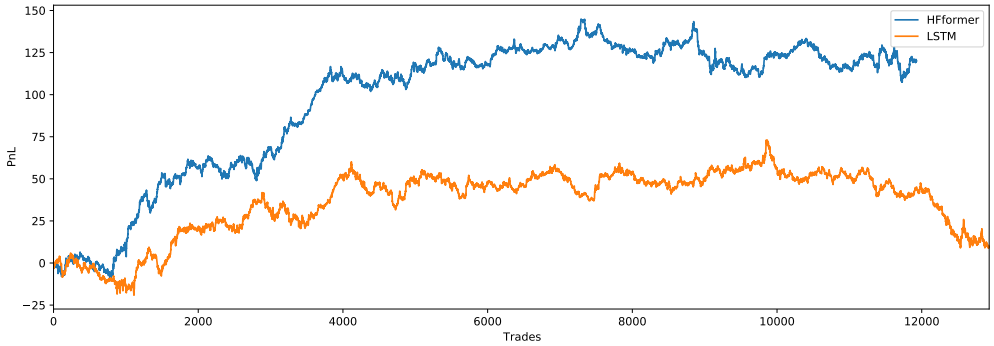

Figure 19: Cumulative PnL - Strategy 2 - HFformer vs. LSTM.

From Table 1, we see that the correlation between adjacent signals is high but not 1. This allows exploring the option of adding more signals and reducing the number of potentially unprofitable trades by only placing trades when the same signal is exhibited for more models.

	PnL	**Signal - 26**	**Signal - 28**	**Signal - 30**
PnL	1			
Signal - 26	0.002603	1		
Signal - 28	0.007165	0.938949	1	
Signal - 30	-0.002530	0.852447	0.867870	1

Table 1: Correlation table for PnL and signals for forecast horizons of 26, 28, and 30 ticks - HFformer.

	PnL	**Signal - 26**	**Signal - 28**	**Signal - 30**
PnL	1			
Signal - 26	0.000514	1		
Signal - 28	0.005031	0.968979	1	
Signal - 30	0.001102	0.972374	0.970612	1

Table 2: Correlation table for PnL and signals for forecast horizons of 26, 28, and 30 ticks - LSTM.

Strategy 3 results in a final PnL of 138.78 USDT and a total of 10,976 trades for the HFformer. By adding two signals, the cumulative PnL has improved by 15.61% compared to strategy 2. As expected, the number of trades decreased by 7.98% compared to strategy 2 and 20.13% compared to strategy 1. Strategy 3 results in a final PnL of 85.71 USDT and a total of 12,226 trades for the LSTM. By adding two signals, the cumulative PnL has improved nine-fold. As expected, the number of trades decreased by 5.46% compared to strategy 2 and 11.04% compared to strategy 1.

Comparing Table 3 and Table 4, we notice that the signals for different forecasting horizons of the HFformer are less correlated than the signals generated by the LSTM. Therefore when HFformer signals are combined, we get fewer trades, but these trades exhibit stronger buy or sell signals. As a result, the HFformer signals, on average, result in a higher PnL than the LSTM signals (cf. Figure 19 and Figure 20).

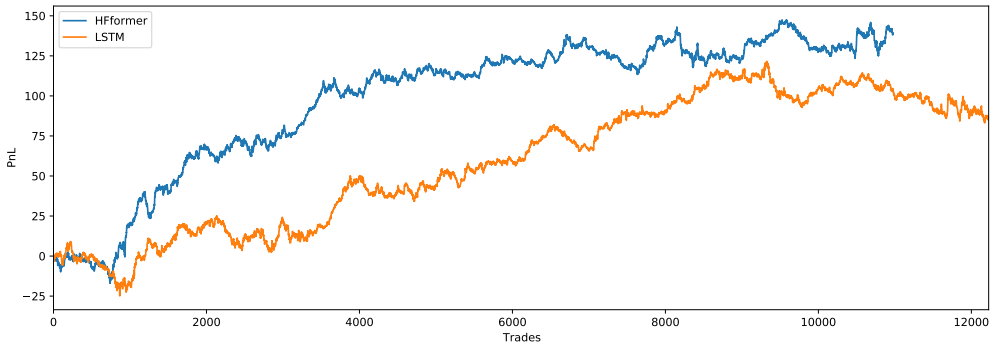

Figure 20: Cumulative PnL - Strategy 3 - HFformer vs. LSTM.

	PnL	Signal - 26	Signal - 27	Signal - 28	Signal - 29	Signal - 30
PnL	1					
Signal - 26	0.009710	1				
Signal - 27	0.013158	0.943009	1			
Signal - 28	0.004809	0.860272	0.877412	1		
Signal - 29	0.014907	0.923525	0.932065	0.858861	1	
Signal - 30	0.010281	0.914158	0.919749	0.861871	0.912567	1

Table 3: Correlation table for PnL and signals for forecast horizons of 26, 27, 28, 29, and 30 ticks - HFformer.

	PnL	Signal - 26	Signal - 27	Signal - 28	Signal - 29	Signal - 30
PnL	1					
Signal - 26	0.007214	1				
Signal - 27	0.011843	0.971106	1			
Signal - 28	0.008985	0.974401	0.972183	1		
Signal - 29	0.007163	0.957233	0.956653	0.953691	1	
Signal - 30	0.010961	0.965258	0.968503	0.964354	0.955223	1

Table 4: Correlation table for PnL and signals for forecast horizons of 26, 27, 28, 29, and 30 ticks - LSTM.

We assess HFformer's performance using 1 to 11 trading signals (cf. Figure 21). The predictions are made for horizon 25 and signals are added progressively by an increment of 1 from each side. We notice an improvement in the cumulative PnL when adding signals. However, after 7 signals the cumulative PnL stagnates. Therefore, for the following experiments we will be using 7 signals.

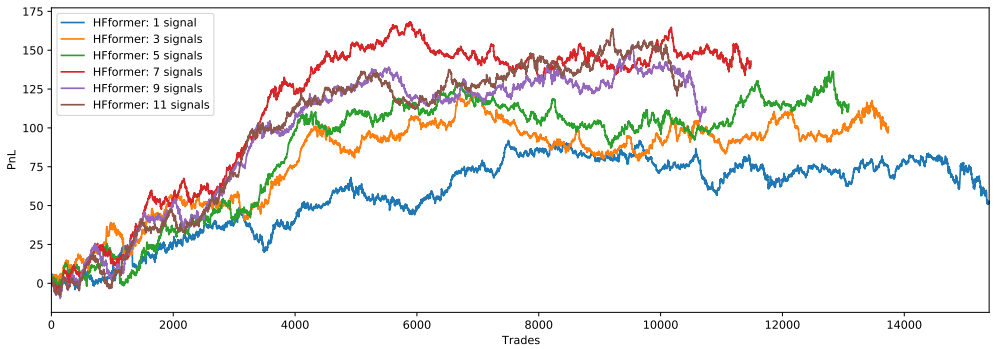

Figure 21: Cumulative PnL - Main horizon 25 - HFformer.

5.9.3 More on Trading Signal Aggregation

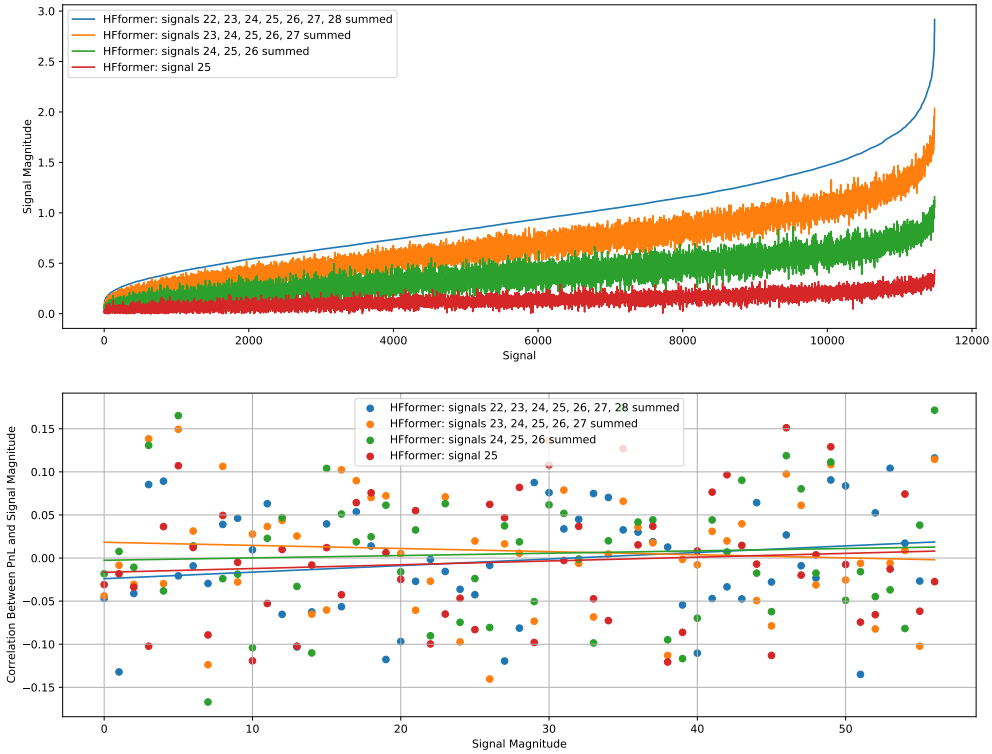

Figure 22: Trade signal magnitude (top) and correlation between PnL and trade signal magnitude (bottom) - Trade sizing - Main forecasting horizon 25 with 1, 3, 5 and 7 signals - HFformer.

From Figure 22, we notice that as we aggregate more signals through summation and then take the absolute value of this sum, the correlation between the PnL and the aggregated signals' magnitude increases (except for the case with five aggregated signals plotted in orange). To perform this comparison, we compute the magnitudes of the aggregated signals and then sort the signals in ascending order. We then compute the correlation between the signal magnitudes and the PnL by batches of 200 observations. Based on these observations, we can improve the trading strategies presented above by using aggregated sums of signals instead of only focusing on the sign of the signals. Additionally, we can disregard trading signals below a certain minimum threshold.

5.9.4 Trade Sizing

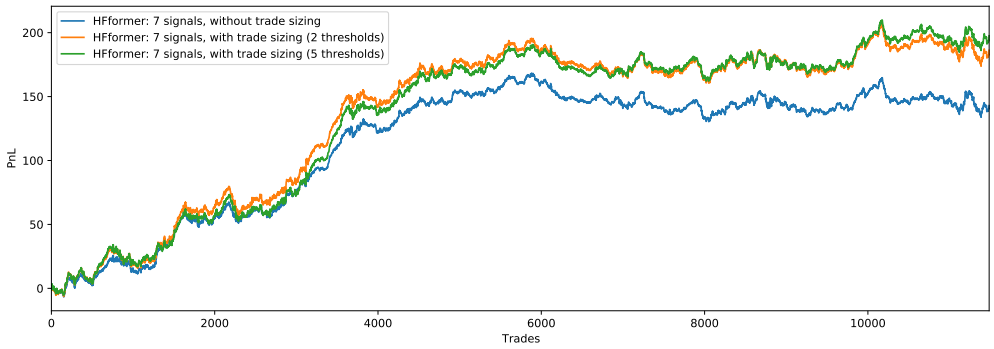

Figure 23: Cumulative PnL - Trade sizing - Main forecasting horizon 25, 7 signals, and 2 and 5 trade sizing thresholds - HFformer.

Finally, we assess HFformer's performance using trade sizing (cf. Figure 23). The trade sizing is done depending on the absolute value of the sum of the trade signals. We use two sets of thresholds:

- 2 thresholds: above the highest threshold 0.15 BTC is traded, 0.1 BTC for the medium threshold, and 0.05 BTC otherwise
- 5 thresholds: above the highest threshold, 0.15 BTC is traded, 0.125 BTC for the second highest threshold, 0.1 BTC for the third highest threshold, 0.075 for the fourth highest threshold, 0.05 for the fifth highest threshold, and 0.025 BTC otherwise

The thresholds above were chosen by observing the distribution of outputted signals using training data.

For this experiment, we use 7 signals with a main forecasting horizon of 25 ticks. We notice an increase in the cumulative PnL when using trade sizing. The standard deviation of the cumulative PnL increases from 0.49 USDT without trade sizing to 0.58 USDT with trade sizing using 2 thresholds. However, when the number of thresholds for trade sizing is increased to 5, the PnL does not increase significantly, but the standard deviation of the PnL drops to 0.41 USDT. A lower standard deviation of the PnL will yield a better Sharpe ratio long-term and reduce the volatility of the high-frequency strategy.

Moreover, from Figure 24, we notice that the ratio of trades with a positive PnL over the total number of trades (i.e., the ratio of winning trades) increases as the magnitude of the trading signal increases in line with previous observations. Therefore, one proposed improvement to the trading strategy is to increase the quantity traded proportionally to the strength of the trading signal.

Finally, we complement the existing trading strategy from Figure 23 with a minimal threshold requirement for a trade to occur. The minimal threshold is set using training data observations. As previously, we compute the sum of trading signals from the models and take the absolute value. The minimal trading threshold is proportional to the number of signals. From Figure 25 we notice an improvement in the cumulative PnL by 4.53% and a decrease in the number of trades from 11,485 to 8,851.

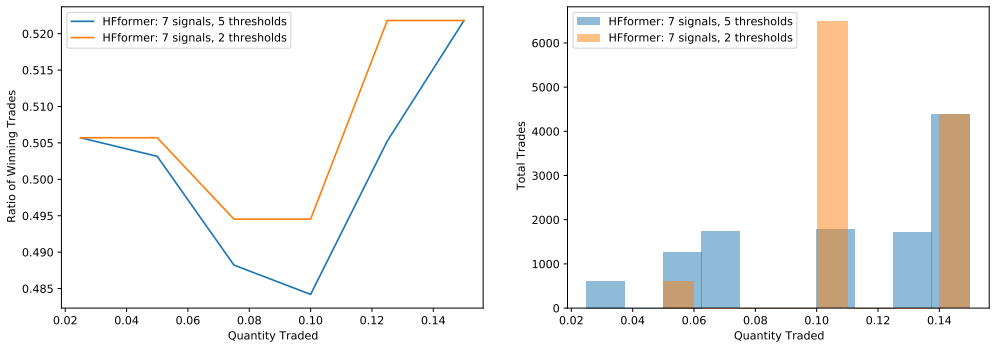

Figure 24: Wining trades ratio - Trade sizing - Main forecasting horizon 25, 7 signals, and 2 and 5 trade sizing thresholds - HFformer.

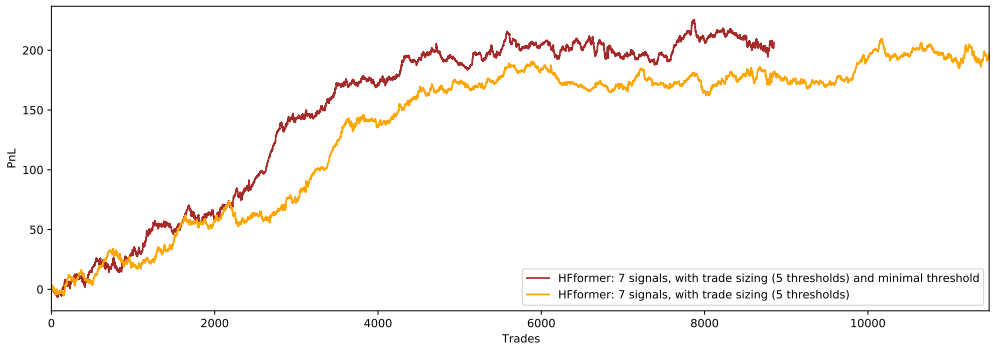

Figure 25: Cumulative PnL - Trade sizing with minimal threshold - Main forecasting horizon 25, 7 signals, and 5 trade sizing thresholds - HFformer.

6 Legal and Ethical Considerations

The strategies and models presented in this paper are purely for scientific research and curiosity. Algorithmic trading may result in capital loss and requires continuous risk monitoring. The strategies should first be paper traded, and various risks should be assessed (e.g., price, liquidity, compliance, and regulatory risks) before being deployed in live trading. Simplifications to the trading environment have been made during backtesting for this paper. Therefore, results may vary significantly if the models are deployed live. Moreover, machine learning methods are evaluated based on classification and regression metrics without rigorous mathematical proofs, unlike statistical methods. As a result, machine learning methods' performance can significantly vary depending on the input data and could lead to unexpected model outputs.

Algorithmic trading is regulated by financial authorities (e.g., Financial Industry Regulatory Authority and Security Exchange Commission for US securities). Regulatory risks need to be considered and monitored when trading in financial markets. The trading strategies need to be reviewed and monitored to be in accordance with continuously changing market regulations.

7 Limitations and Future Lines of Research

The HFformer and LSTM results were achieved using a simplified backtesting environment, which did not account for factors including:

- the impact of the executed order on the market
- the actual execution price; this is set to the weighted midprice in our model
- the available BTC quantities; the executed quantity is 0.1 BTC
- any connection issues; the data was fed into the model at the same rate
- trading fees; the trading fee was set to zero, which is a realistic assumption for the BTC-USDT pair on Binance. However, trading on traditional exchanges usually requires paying a transaction fee
- a longer backtesting period; the backtesting period is 2 days

The following future lines of research can be undertaken:

- Improve the LOB snapshot pre-processing pipeline to reduce the noisiness of the data with automated feature selection by using autoencoders [32]
- Perform a more extensive performance assessment of the HFformer on large forecast horizons and use altcoin trading pairs such as ETH-USDT
- Implement the HFformer with other types of Attention modules such as auto-correlation Attention [15]
- Implement a more realistic backtesting environment that accounts for the impact of the placed order and emulates the activity of other participants in the market to assess the performance of the HFformer
- Test the HFformer's forecasting performance on traditional financial data by using LOB data from other papers focused on LOB deep learning models for midprice forecasting of traditional assets[10]

8 Conclusion

This paper studied the LSTM, Transformer, Autoformer, FEDformer, TFT, and HFformer deep learning architectures for high-frequency FTS forecasting.

Through experimentation, we combined various components of multiple Transformer architectures to form the HFformer, a Transformer-like architecture adapted for HFT. When testing the LSTM and HFformer models, we achieved a higher R^2 score than the other deep learning architectures for log-returns forecasting from 1 to 30 ticks ahead. Moreover, the LSTM and HFformer models achieved similar performance for classification tasks. Finally, the LSTM and HFformer models were backtested on different trading strategies involving 1, 3, and 5 trade signals. As a result, it was found that using more than 1 trade signal decreases the number of trades and increases the cumulative PnL of a long-short trading strategy.

The HFformer, which uses the multi-head Attention mechanism, generates long and short trade signals that result in a more balanced trading strategy than the LSTM. Moreover, the trade signals generated by the HFformer for different forecasting horizons are less correlated than the LSTM signals. As a result, a trading

strategy that combines multiple trade signals around a given forecasting horizon was proposed. This trading strategy reduces the false positive trade signals by only engaging in trades when all signals are of the same sign, which results in higher cumulative PnLs and fewer trading fees.

Finally, the proposed trading strategies were complemented with trade sizing to improve the cumulative PnL. By using different trading quantities based on the strength of the trading signal, the cumulative PnL increased, and in some cases, the volatility of the trading strategy decreased. Additionally, ignoring trade signals below a predefined minimal threshold reduced the total number of trades and positively contributed to the average trade PnL.

Although these improvements and strategies have been backtested on a large amount of BTC-USDT LOB data collected over 2 days and a month after the training and validation data, these methods may yield different results when trading another cryptocurrency pair or financial asset. Additionally, machine learning methods may sometimes be less generalizable than traditional statistical methods as the machine learning methods are data-driven.

References

[1] Box GE, Jenkins GM, Reinsel GC, Ljung GM. Time series analysis: forecasting and control. John Wiley & Sons; 2015. pages 1

[2] Engle RF. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. Econometrica: Journal of the econometric society. 1982:987-1007. pages 1

[3] Rumelhart DE, Hinton GE, Williams RJ. Learning internal representations by error propagation. California Univ San Diego La Jolla Inst for Cognitive Science; 1985. pages 1

[4] Hochreiter S, Schmidhuber J. Long short-term memory. Neural computation. 1997;9(8):1735-80. pages 1

[5] LeCun Y, Bengio Y, Hinton G. Deep learning. nature. 2015;521(7553):436-44. pages 1

[6] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. Advances in neural information processing systems. 2017;30. pages 1, 5

[7] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:14090473. 2014. pages 1

[8] Liu Z, Lin Y, Cao Y, Hu H, Wei Y, Zhang Z, et al. Swin transformer: Hierarchical vision transformer using shifted windows. In: Proceedings of the IEEE/CVF International Conference on Computer Vision; 2021. p. 10012-22. pages 1

[9] Tsay RS. Analysis of financial time series. John wiley & sons; 2005. pages 1

[10] Kolm PN, Turiel J, Westray N. Deep Order Flow Imbalance: Extracting Alpha at Multiple Horizons from the Limit Order Book. Available at SSRN 3900141. 2021. pages 1, 2, 17

[11] Qin Y, Song D, Chen H, Cheng W, Jiang G, Cottrell G. A dual-stage attention-based recurrent neural network for time series prediction. arXiv preprint arXiv:170402971. 2017. pages 1, 2

[12] Cartea Á, Jaimungal S, Penalva J. Algorithmic and high-frequency trading. Cambridge University Press; 2015. pages 2

[13] Sezer OB, Gudelek MU, Ozbayoglu AM. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. Applied soft computing. 2020;90:106181. pages 2

[14] Pascanu R, Mikolov T, Bengio Y. On the difficulty of training recurrent neural networks. In: International conference on machine learning. PMLR; 2013. p. 1310-8. pages 2

[15] Xu J, Wang J, Long M, et al. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. Advances in Neural Information Processing Systems. 2021;34. pages 2, 3, 7, 17

[16] Zhou T, Ma Z, Wen Q, Wang X, Sun L, Jin R. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. arXiv preprint arXiv:220112740. 2022. pages 2, 3

[17] Angel J. Retail Investors Get a Sweet Deal: The Cost of a SIP of Stock Market Data. Available at SSRN 3268916. 2018. pages 2

[18] Hautsch N. Modelling irregularly spaced financial data: theory and practice of dynamic duration models. vol. 539. Springer Science & Business Media; 2011. pages 2

[19] Ariyo AA, Adewumi AO, Ayo CK. Stock price prediction using the ARIMA model. In: 2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation. IEEE; 2014. p. 106-12. pages 2

[20] Franses PH, Van Dijk D. Forecasting stock market volatility using (non-linear) Garch models. Journal of forecasting. 1996;15(3):229-35. pages 2

[21] Fischer T, Krauss C. Deep learning with long short-term memory networks for financial market predictions. European Journal of Operational Research. 2018;270(2):654-69. pages 2

[22] Lim B, Zohren S. Time-series forecasting with deep learning: a survey. Philosophical Transactions of the Royal Society A. 2021;379(2194):20200209. pages 2

[23] Gao K, Luk W, Weston S. High-Frequency Trading and Financial Time-Series Prediction with Spiking Neural Networks. Wilmott. 2021;2021(113):18-33. pages 2

[24] Wang Y, Huang M, Zhu X, Zhao L. Attention-based LSTM for aspect-level sentiment classification. In: Proceedings of the 2016 conference on empirical methods in natural language processing; 2016. p. 606-15. pages 2

[25] Wu N, Green B, Ben X, O'Banion S. Deep transformer models for time series forecasting: The influenza prevalence case. arXiv preprint arXiv:200108317. 2020. pages 2

[26] Kazemi SM, Goel R, Eghbali S, Ramanan J, Sahota J, Thakur S, et al. Time2vec: Learning a vector representation of time. arXiv preprint arXiv:190705321. 2019. pages 2

[27] Li S, Jin X, Xuan Y, Zhou X, Chen W, Wang YX, et al. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. Advances in Neural Information Processing Systems. 2019;32. pages 3

[28] Lim B, Arik SO, Loeff N, Pfister T. Temporal fusion transformers for interpretable multi-horizon time series forecasting. arXiv preprint arXiv:191209363. 2019. pages 3

[29] Ba JL, Kiros JR, Hinton GE. Layer normalization. arXiv preprint arXiv:160706450. 2016. pages 6

[30] Irie K, Zeyer A, Schlüter R, Ney H. Language modeling with deep transformers. arXiv preprint arXiv:190504226. 2019. pages 8

[31] Loshchilov I, Hutter F. Decoupled weight decay regularization. arXiv preprint arXiv:171105101. 2017. pages 9

[32] Han K, Wang Y, Zhang C, Li C, Xu C. Autoencoder inspired unsupervised feature selection. In: 2018 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE; 2018. p. 2941-5. pages 17

The following Github repositories were used to build the deep learning models:

- LSTM: https://github.com/lkulowski/LSTM_encoder_decoder
- Transformer: https://github.com/AIStream-Peelout/flow-forecast
- Autoformer and FEDformer: https://github.com/cure-lab/DLinear
- Spiking activation: https://github.com/nengo/pytorch-spiking

9 Appendix

The training of the deep learning models was done on Google Colab[4] using an Nvidia P100 GPU. The training LOB data for the USDT-BTC trading pair collected from Binance's API can be found below:

- raw LOB data: https://drive.google.com/drive/folders/1GXXiVyXXCXenNsGWRAmMb Xd1Xvf-lZq5?usp=sharing
- processed LOB data: https://drive.google.com/drive/folders/1GVJ050lVeS6vFjZ9CER YkL5W5YhKOLHw?usp=sharing

The hyperparameters that were used to train the LSTM, Transformer, Autoformer, FEDformer, and HF-former models were found through grid search and are listed below:

[4]The Google Colab platform can be found at https://colab.research.google.com/notebooks/.

Model	Layers	Activation Function	Optimizer	Loss Function	Batch Size
LSTM	5 layers size 16	PReLU	AdamW with 0.001 learning rate	MSE	64
Transformer	1 encoder and 1 decoder both size 128 with 8 Attention heads	PReLU	AdamW with 0.0001 learning rate	MSE	64
Autoformer	2 encoders and 2 decoders both of size 128 with 8 Attention heads, label length 50, and factor 3	GELU	AdamW with 0.0002 learning rate	MSE	256
FEDformer	2 encoders and 2 decoders both of size 128 with 8 Attention heads, label length 50, and Legendre base	GELU	AdamW with 0.0005 learning rate	MSE	256
HFformer	1 Transformer encoder and 1 linear decoder both size 64 with 6 Attention heads	Spiking with PReLU	AdamW with 0.04 learning rate	MSE	256

Table 5: LSTM, Transformer, Autoformer, FEDformer, and HFformer hyperparameters.