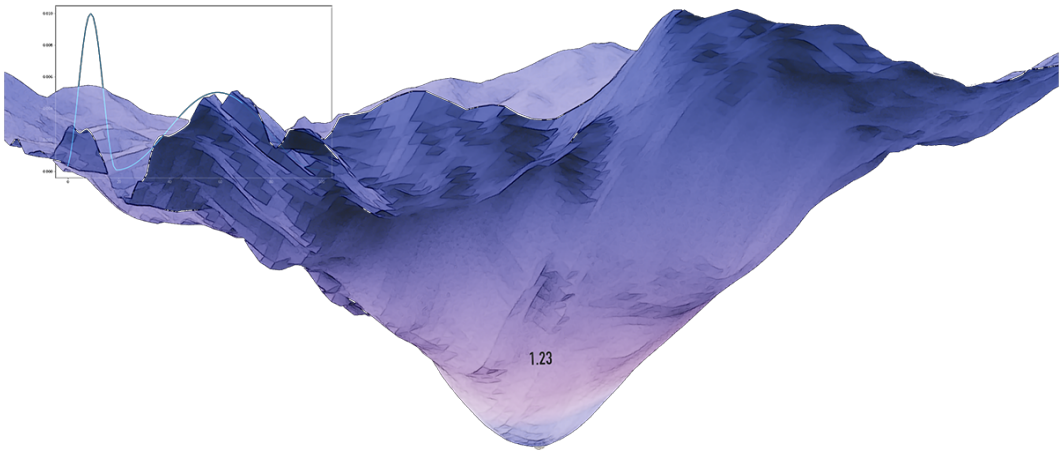
# INSIDER INFORMATION IS NOT THE ONLY EDGE WHEN FORECASTING FINANCIAL MARKETS

**Nicholas Erup Larsen***
s224175@dtu.dk / nicholaseruplarsen@gmail.com
BSc in Artificial Intelligence & Data, DTU
Founder & CEO of Erup Capital




## ABSTRACT

Deep learning models, particularly RNNs, LSTMs, and now Transformers, have shown remarkable capabilities for learning sequential dependencies in data with known structure (like language). However, as shown in the time series literature, their performance in forecasting financial markets is surprisingly suboptimal compared to simpler baselines, reinforcing the belief that these are fundamentally random (EMH). However, if financial datasets only contain some amount of noise, then there must exist a filter which when applied can enhance these models' ability to learn the underlying signal. This thesis shows that applications of FFT-based filtering to financial datasets significantly enhances the ability of LSTMs to generalize to the future, thereby also disproving all colloquially known forms of the Efficient Market Hypothesis (EMH).

*Thank you to my supervisor, **Tommy Sonne Alstrøm**, for his guidance and support.

# Contents

# 1 Introduction

## 1.1 The Case for Efficient Market Hypothesis (EMH)

The Efficient Market Hypothesis (EMH) stands as a cornerstone of modern financial theory, positing that financial asset prices fully reflect all available information. The theoretical foundation can be traced back to the pioneering work of Louis Bachelier Bachelier (1900)[1], who first proposed a mathematical framework for understanding price fluctuations in speculative markets.

Bachelier's insight was to model price movements using the mathematics of random processes, specifically Brownian motion. While he did not explicitly set out to prove market efficiency, his fundamental assumption was that market prices reflect the aggregate expectations of all participants, making future price movements inherently unpredictable. In the introduction to his thesis, he claims:

> INTRODUCTION
>
> The influences which determine the movements of the Stock Exchange are innumerable. Events past, present or even anticipated, often showing no apparent connection with its fluctuations, yet have repercussions on its course. Beside fluctuations from, as it were, natural causes, artificial causes are also involved. The Stock Exchange acts upon itself and its current movement is a function not only of earlier fluctuations, but also of the present market position.
>
> The determination of these fluctuations is subject to an infinite number of factors: it is therefore impossible to expect a mathematically exact forecast. Contradictory opinions in regard to these fluctuations are so divided that at the same instant buyers believe the market is rising and sellers that it is falling.
>
> Undoubtedly, the Theory of Probability will never be applicable to the movements of quoted prices and the dynamics of the Stock Exchange will never be an exact science.
>
> However, it is possible to study mathematically the static state of the market at a given instant, that is to say, to establish the probability law for the price fluctuations that the market admits at this instant. Indeed, while the market does not foresee fluctuations, it considers which of them are more or less probable, and this probability can be evaluated mathematically.
>
> Up to the present day, no investigation into a formula for such an expression appears to have been published: that will be the object of this work.
>
> ...

Bachelier's model assumed that price changes, $dP$, over an infinitesimal time interval, $dt$, are drawn from a normal distribution with zero mean, implying that the best forecast for tomorrow's price is today's price (also known as the Martingale property). Mathematically, this can be represented as

$$\Delta P_t \sim N(0, \sigma^2 \Delta t) \quad \text{and} \quad E[P_{t+dt} - P_t | P_t] = 0$$

The empirical evidence for this assertion is 5 years of government bond prices (French 3% Rente) from 1894 to 1898, which roughly tracks with his pricing formula.

In any case, his mathematical formalization of unpredictable price movements and "fair game" option pricing was the first to formalize the concept of Geometric Brownian Motion (GBM), which became the bedrock of the famous landmark Black & Scholes (1973) option pricing model. This model extended Bachelier's work crucially such that stock prices couldn't be negative, described mathematically as:

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

Here, $dS_t$ is the change in the stock price, $\mu$ is the expected rate of return (drift), $\sigma$ is the volatility (standard deviation of returns), $dt$ is an infinitesimal time interval, and $dW_t$ is a Wiener process increment (representing the random shock), satisfying $dW_t \sim N(0, dt)$. This model implies that logarithmic returns, $\ln(S_t/S_{t-\Delta t})$, are normally distributed.

[1] Bachelier's 1900 doctoral thesis, *Théorie de la Spéculation*, was later translated to English and popularised in collections such as Davis & Etheridge (2006).

A common critique, and often arbitraged opportunity, in the assumption of the Black-Scholes framework is that of constant volatility $\sigma$, where realized volatility often doesn't track with what is implied by the model. However, as of yet, few challenge the assumption of GBM itself.

Nevertheless, the model's success in pricing options profitably lent substantial credence to the underlying random walk hypothesis for asset prices, or at least as a practical approximation for market makers. Again, the fundamental assumption driving the formula relies on the principle of no-arbitrage: if risk-free profits could be made due to mispricing, arbitrageurs would quickly eliminate such opportunities, driving prices back to their "efficient" or "fair value" levels.

Lastly, Eugene Fama (1970) later systematized these concepts, categorizing market efficiency into three distinct forms, each defined by the set of information[2] assumed to be incorporated into prices:

**Weak-Form Efficiency:** Current asset prices fully reflect all information contained in past market data, such as historical prices and trading volumes. Consequently, technical analysis, which seeks to identify patterns based on past price movements to predict future prices, is a fool's errand. If this form holds, Fama (seriously, unironically) formalizes this as:

$$\mathbb{E}[R_{i,t+1}|\Phi_t^W] = \mathbb{E}[R_{i,t+1}]$$

where $R_{i,t+1}$ is the return of asset $i$ at time $t+1$, and $\Phi_t^W$ are historical price movements.

**Semi-Strong Form Efficiency:** Asset prices adjust rapidly and unbiasedly to all publicly available information. This includes not only past prices but also company announcements, economic news, analyst reports, and other public data. If true, neither technical nor fundamental analysis (based on public information) can consistently yield risk-adjusted excess returns. Here,

$$\mathbb{E}[R_{i,t+1}|\Phi_t^{SS}] = \mathbb{E}[R_{i,t+1}]$$

where $\Phi_t^{SS}$ includes all publicly available information. In other words, insider information IS the only edge when forecasting financial markets.

**Strong-Form Efficiency:** Asset prices reflect all information, both public and private (insider information). This is the strictest form and implies that no one can consistently achieve superior returns to that of the general drift of the market, due to its stochastic nature. In this case,

$$\mathbb{E}[R_{i,t+1}|\Phi_t^S] = \mathbb{E}[R_{i,t+1}]$$

where $\Phi_t^S$ encompasses all information.

So these all read as: The expected future (cumulative) return of an asset GIVEN (conditioned on) all information is equal to the expected the current return of the asset, i.e. fundamentally unpredictable.

If price increments were not random (e.g., drawn from a normal distribution with mean zero for log returns over short intervals), but instead exhibited predictable skewness or kurtosis beyond that accounted for by risk premia, this would signal an exploitable inefficiency. For instance, if the distribution of log returns were consistently skewed to the right, it would suggest a persistent tendency for positive surprises, an opportunity that rational investors would quickly learn to anticipate and exploit, thereby neutralizing the skew.

So in summary, the argument goes: If a market then is populated by rational, profit-maximizing agents who actively compete to uncover and exploit any mispricing (no-arbitrage principle), information is almost instantaneously priced in. Any discernible pattern or predictability that could lead to abnormal profits would be arbitraged away almost instantaneously. Therefore, price changes must be unforecastable, responding only to the arrival of new, unexpected information.

Empirically, this also looks to be true:

[plot of most returns being normal distributed]

[2]Not the same as Shannon information

### 1.2 The Case Against the Efficient Market Hypothesi (EMH)

While the Efficient Market Hypothesis (EMH) offers an elegant, simple, and solved way to think about markets, its strong assertions about information incorporation and price unpredictability have faced persistent challenges from both theoretical and empirical standpoints. The hypothesis, particularly in its semi-strong and strong forms, rests on assumptions of near-perfect rationality, costless information, and instantaneous arbitrage—conditions that one can argue are not yet fully met in real-world financial markets. This section outlines key arguments and observations that question the universal applicability of EMH and suggest the potential for discoverable patterns in price movements.

My central assertion, which this thesis aims to explore through the lens of self-supervised learning and information theory, is that:

> Price movements, particularly in the short-term, often exhibit non-random characteristics and patterns from previous price movements, implying a degree of predictability.

This is not to say that markets are grossly inefficient or that arbitrage opportunities are abundant and easily exploitable, but rather that the strict informational efficiency posited by EMH may be an oversimplification.

One foundational critique stems from the very nature of price discovery. The EMH implies that new information instantaneously translates into a new, correct ”fair value.” However, the concept of a singular, universally agreed-upon ”fair value” is itself debatable, particularly in markets influenced by speculation, heterogeneous beliefs, and evolving narratives (e.g., the valuation of growth stocks, cryptocurrencies, or assets subject to significant sentiment swings Shiller (2000)).[3] The process of price adjustment to new information is often more akin to an *auction market* or a ”tug of war,” where prices may oscillate, trend, or exhibit cyclical patterns as market participants gradually digest information and converge, or fail to converge, towards a new equilibrium O'Hara (1995). If the path to this new (potentially transient) equilibrium is not random, then predictability arises.

Furthermore, the assumption that market participants are perfectly rational and that arbitrage is perfectly effective is challenged by the field of behavioral finance Kahneman & Tversky (1979); Thaler (1993). Documented psychological biases such as herd behavior, anchoring, overconfidence, and loss aversion can lead to systematic deviations from rational pricing. For example:

Markets may overreact to dramatic news or underreact to gradual streams of information, leading to momentum effects (prices continuing in a trend) or reversals (prices correcting after an overshot) Jegadeesh & Titman (1993); Fama (1998).

And even if mispricings exist, rational arbitrageurs may be constrained by transaction costs, risks (e.g., noise trader risk, synchronization risk), or capital limitations, preventing them from fully correcting these inefficiencies Shleifer (2000).

Consider the analogy of human height distribution. If an alien observer only saw the aggregate distribution of adult heights, they might conclude it's (roughly) a normal distribution and therefore ”random.” However, we know that underlying factors, such as parental height, genetics, and nutrition, provide significant predictive power for an individual's height. Similarly, while aggregate market returns over certain periods might superficially resemble a random walk or a normal distribution (as shown in Figure 1), this does not preclude the existence of underlying, learnable patterns or dependencies that govern individual price transitions.

The very premise of EMH, that a perfectly traded market leads to a random walk of price changes rather than, for instance, a more direct path to a new equilibrium after news, is also open to question. Why should the default assumption for the price discovery process, i.e. the dynamics between buyers and sellers reacting to information and to each other, be inherently stochastic rather than exhibiting temporary, exploitable structure? Could it be that in the process of converging towards a new ”fair value” following new information, the collective psychology, strategic interactions of traders, or

[3]The concept of ”fair value” becomes particularly vague with assets like ”TSLA or FARTCOIN.” While Fama's original work predates many more modern market phenomena, the challenge of defining and agreeing upon intrinsic value persists and is amplified by rapid information flow and social media influence.

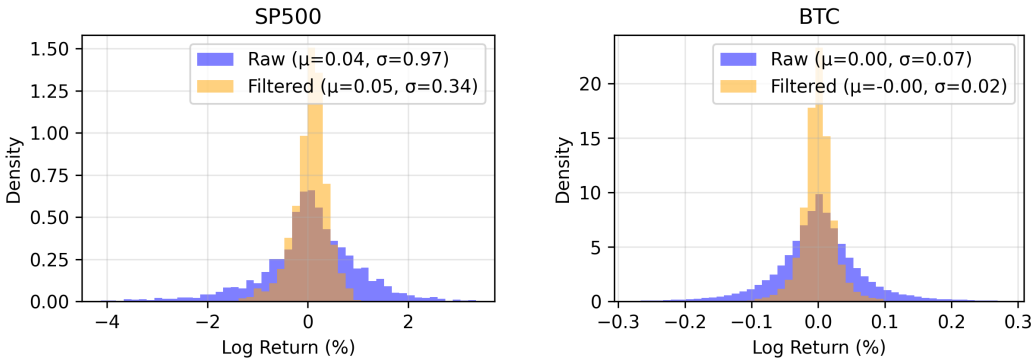



Figure 1: Log Returns Distribution for S&P500 (daily) and Bitcoin (minutely), window size 15

feedback loops inherent in market dynamics cause prices to move in a somewhat predictable fashion, at least for short durations, until disagreement settles or a new consensus forms?

This brings us to Fama's conditioning information set, $\Phi_t$, in $\mathbb{E}[R_{i,t+1}|\Phi_t] = \mathbb{E}[R_{i,t+1}]$. The claim that $\Phi_t$ truly encompasses "all available information" (or even all publicly available information for semi-strong form) and that this vast, complex, and often ambiguous information is instantaneously and perfectly processed by all market participants into a single efficient price is an extraordinarily strong one. It verges on assuming a collective omniscience and processing power that seems implausible in practice.

Therefore, the case against EMH is not necessarily a declaration of rampant market irrationality, but rather an acknowledgment that markets are complex adaptive systems populated by boundedly rational agents, where information diffusion is imperfect, and the price discovery process itself can generate temporary patterns.

My hypothesis is that such inefficiencies and patterns will persist as long as the price adjustment mechanism is not a literal instantaneous jump from one equilibrium to another but a process unfolding over time. Self-supervised learning techniques, by their nature, are designed to uncover precisely these types of inherent structures and dependencies within sequential data.

### 1.3 Bridging to Information Theory: Mutual Information and Entropy

The critique of EMH, particularly the assertion that price movements are not entirely unpredictable and may depend on past movements, can be rigorously framed using concepts from information theory, pioneered by Claude ShannonShannon (1948). Information theory provides a mathematical framework to quantify uncertainty and information content, offering a lens through which to analyze the randomness (or lack thereof) in financial time series.

At the core of this perspective is **Shannon Entropy**. For a discrete random variable $X$ with possible outcomes $x_i$ and probabilities $p(x_i)$, the entropy $H(X)$ is defined as:

$$H(X) = -\sum_i p(x_i) \log_2 p(x_i)$$

Entropy measures the average uncertainty or "surprise" associated with the outcomes of $X$. A higher entropy indicates greater randomness and less predictability. If financial price changes truly follow a random walk as implied by EMH (specifically, if log returns are independent and identically distributed, e.g., from a normal distribution with constant parameters), then the conditional entropy of a future price change, given past price changes, should be maximized and equal to its unconditional entropy. That is, past information offers no reduction in uncertainty about the future.

If the probabilities of outcomes are balanced (e.g., a 0.5 probability of an "up" tick or "down" tick of a certain magnitude, akin to a binomial distribution converging to a normal distribution for many steps), entropy is maximized. If, however, probabilities are skewed or if outcomes are dependent on previous outcomes, entropy is reduced, implying that some information or predictability is present.

The EMH, particularly its weak form, posits that historical price information $\Phi_t^W$ (e.g., $P_t, P_{t-1}, \dots$) contains no information that can be used to predict future price changes $\Delta P_{t+1}$. In information-theoretic terms, this implies that the **Mutual Information (MI)** between past price changes and future price changes should be zero (or statistically insignificant). Mutual information $I(X;Y)$ between two random variables $X$ and $Y$ measures the amount of information obtained about one random variable through observing the other. It quantifies the reduction in uncertainty about $Y$ given knowledge of $X$:

$$I(X;Y) = H(Y) - H(Y|X)$$

If $I(\Delta P_{t+1}; \Phi_t^W) > 0$, it means that past price information $\Phi_t^W$ reduces the uncertainty about the next price change $\Delta P_{t+1}$, contradicting the weak-form EMH.

If we think of neural networks as compressors, then a sufficiently powerful predictive model is implicitly learning to compress the data. If a model then predicts the next elements in a sequence with significantly higher than 50% accuracy, it means it has identified patterns, redundancies, or dependencies from the preceding sequence. If a time series of price changes can be compressed to a representation smaller than what would be expected if it were truly random, then

Therefore, this thesis will leverage self-supervised learning not just to make forecasts, but as a tool to estimate the degree of structure and dependency within financial time series. By training models to predict future data from previous parts of itself. The success of such models, quantified by metrics that can be related to information-theoretic measures, can serve as evidence against the strong statistical implications of EMH. If patterns exist, entropy is lower than maximum, mutual information between past and future is non-zero, and the data is, to some extent, compressible and predictable.

## 2 Research Questions

- How do different smoothing methods and their window sizes affect the information content of the data and thereby performance of the model's predictions?
- 3m contains more "signal" than 1m, 5m more than 3m, etc. Why?
- Why does predicting 1h ahead perform worse than 60m (1m/3m/5m granularity) ahead despite the latter having more data? Surely, in an ideal world, the more data, the better?
- Why does the last value in the input sequence contain the most affect predictions so much?

# 3 Theory

## 3.1 Neural Network

A Neural Network (NN), or more specifically an Artificial Neural Network (ANN), is a computational model inspired by the structure and function of biological neural networks in animal brains ANNs are composed of interconnected processing units called neurons or nodes, typically organized in layers: an input layer, one or more hidden layers, and an output layer.

Each connection between neurons has an associated weight, which modulates the signal transmitted through it. Neurons in hidden and output layers also typically have a bias term. A neuron computes a weighted sum of its inputs (plus its bias) and then applies a non-linear activation function to this sum to produce its output. Common activation functions include the sigmoid, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU). Mathematically, the output $a_j$ of a neuron $j$ in a layer can be expressed as:

$$a_j = f\left(\sum_i w_{ji} x_i + b_j\right) \tag{1}$$

where $x_i$ are the inputs from the previous layer (or the raw input data), $w_{ji}$ are the weights connecting neuron $i$ to neuron $j$, $b_j$ is the bias of neuron $j$, and $f(\cdot)$ is the activation function.

Feedforward Neural Networks (FFNNs), where connections do not form cycles, are a common type. Information flows in one direction, from input to output. NNs "learn" by adjusting their weights and biases through a process called training, typically to minimize a predefined error or loss function that quantifies the discrepancy between the network's predictions and the true target values.

## 3.2 Activation Functions

Activation functions are a critical component of neural networks, primarily responsible for introducing non-linearity into the model. As described in Equation 1, each neuron computes a weighted sum of its inputs plus a bias, and then applies an activation function $f(\cdot)$ to this sum. The result of this function becomes the neuron's output, which is subsequently passed as input to neurons in the next layer.

The introduction of non-linearity is essential. Without it, a neural network composed of multiple layers would mathematically collapse into an equivalent single-layer linear model. Such a linear model would be incapable of learning the complex, non-linear relationships often present in real-world data. By enabling these non-linear transformations, activation functions empower neural networks to approximate intricate functions and learn complex patterns from the data. Common examples include the Sigmoid, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU), each possessing distinct characteristics that influence network training and performance.

The **Sigmoid** function squashes its input into a range between 0 and 1, and is defined as:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

The **hyperbolic tangent (tanh)** function is similar to sigmoid but squashes its input into a range between -1 and 1:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$

The **Rectified Linear Unit (ReLU)** is a widely used function that outputs the input directly if it is positive, and zero otherwise:

$$f(x) = \text{ReLU}(x) = \max(0, x) \tag{4}$$

These functions, among others, allow neural networks to learn a rich variety of mappings from inputs to outputs.

## 3.3 Backpropagation

Training a neural network involves two main phases that are executed iteratively: the forward pass and the backward pass.

Forward Pass: Input data, denoted as $\mathbf{x}$, is fed into the network. For each layer $l$, starting from the input layer ($l = 1$) through any hidden layers to the output layer ($l = L$), the computation proceeds as follows. The weighted sum (or pre-activation) $\mathbf{z}^{(l)}$ for layer $l$ is calculated as:

$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \tag{5}$$

where $W^{(l)}$ is the weight matrix, $\mathbf{b}^{(l)}$ is the bias vector for layer $l$, and $\mathbf{a}^{(l-1)}$ is the activation vector from the previous layer (with $\mathbf{a}^{(0)} = \mathbf{x}$ representing the input data). An activation function $f(\cdot)$ (or $f_l(\cdot)$ if layer-specific) is then applied element-wise to produce the activations for layer $l$:

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)}) \tag{6}$$

This process is repeated until the final output of the network, $\hat{\mathbf{y}} = \mathbf{a}^{(L)}$, is obtained. This output is then compared to the true target values $\mathbf{y}$ using a loss function $L(\hat{\mathbf{y}}, \mathbf{y})$ to quantify the network's error.

Backward Pass: The algorithm starts at the output layer and propagates the error signal backward through the network. It calculates the gradient of the loss function with respect to the parameters (weights and biases) of each layer. The chain rule of calculus is fundamental to backpropagation, allowing the error contribution of each parameter to be determined efficiently. For a given weight $w_{ji}$ (denoting the weight connecting neuron $i$ in layer $l-1$ to neuron $j$ in layer $l$), the gradient $\frac{\partial L}{\partial w_{ji}}$ indicates how a small change in $w_{ji}$ affects the total loss $L$. The computation relies on breaking down this derivative. If $a_j^{(l)}$ is the activation of neuron $j$ in layer $l$, and $z_j^{(l)}$ is its weighted input sum (i.e., $z_j^{(l)} = \sum_k w_{jk} a_k^{(l-1)} + b_j^{(l)}$, where the sum is over all neurons $k$ in layer $l-1$ connecting to neuron $j$), the partial derivative of the loss with respect to $w_{ji}$ is expressed via the chain rule as:

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{ji}} \tag{7}$$

The first two terms on the right-hand side, $\frac{\partial L}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}$, combine to form $\frac{\partial L}{\partial z_j^{(l)}}$. This quantity is often denoted as the error signal $\delta_j^{(l)}$ for neuron $j$ in layer $l$. It quantifies how a small change in the pre-activation $z_j^{(l)}$ (the input to the activation function) affects the total loss $L$. The third term, $\frac{\partial z_j^{(l)}}{\partial w_{ji}}$, represents how the weight $w_{ji}$ affects $z_j^{(l)}$, and this simplifies to $a_i^{(l-1)}$, the activation of neuron $i$ from the previous layer $l-1$ (which provided input to $w_{ji}$). Thus, the gradient for a specific weight $w_{ji}$ is compactly expressed as:

$$\frac{\partial L}{\partial w_{ji}} = \delta_j^{(l)} a_i^{(l-1)} \tag{8}$$

Similarly, the gradient for the bias $b_j^{(l)}$ associated with neuron $j$ in layer $l$ is $\frac{\partial L}{\partial b_j^{(l)}} = \delta_j^{(l)}$, since $\frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = 1$.

Backpropagation's effectiveness arises from its efficient, recursive computation of these $\delta_j^{(l)}$ terms. Instead of calculating each gradient component in Equation 7 independently from scratch for every weight (which would involve many redundant computations), backpropagation computes the $\delta_j^{(l)}$ values by propagating error signals backward from the output layer.

For the output layer $L$, $\delta_j^{(L)}$ is calculated directly using the derivative of the loss function with respect to the output activations and the derivative of the output layer's activation function $f_L(\cdot)$:

$$\delta_j^{(L)} = \frac{\partial L}{\partial a_j^{(L)}} f_L'(z_j^{(L)}) \tag{9}$$

For any preceding hidden layer $l$ (from $L-1$ down to 1), the error term $\delta_j^{(l)}$ is computed using the error terms $\delta_k^{(l+1)}$ from the subsequent layer $l+1$:

$$\delta_j^{(l)} = \left( \sum_k \delta_k^{(l+1)} w_{kj}^{(l+1)} \right) f_l'(z_j^{(l)}) \tag{10}$$

where $w_{kj}^{(l+1)}$ is the weight connecting neuron $j$ of layer $l$ to neuron $k$ of layer $l+1$ (note: this $w_{kj}^{(l+1)}$ carries the error from $\delta_k^{(l+1)}$ back to $z_j^{(l)}$), and $f_l'(z_j^{(l)})$ is the derivative of the activation function of neuron $j$ in layer $l$. The sum effectively weights the error signals from layer $l+1$ by the strengths of the connections $w_{kj}^{(l+1)}$ that transmit these signals back to neuron $j$.

This recursive calculation (Equations 9 and 10) allows the error signals $\delta^{(l)}$ to be efficiently propagated backward. Each $\delta_j^{(l)}$ value is computed once and then reused to calculate the gradients for all weights $w_{ji}$ feeding into neuron $j$ (via Equation 8) and for the bias $b_j^{(l)}$, as well as to propagate errors to the preceding layer $l-1$. This dynamic programming characteristic avoids recomputing shared parts of the chain rule expressions. While a naive calculation of all gradients might scale with the number of parameters multiplied by the cost of a forward pass, backpropagation's computational cost is typically proportional to that of a single forward pass, making the training of deep networks feasible. It ensures that the influence of each parameter on the total error is determined systematically.

Once all gradients are computed, an optimization algorithm uses them to update the weights and biases, typically by taking a step in the negative direction of the gradient. This process is repeated for many iterations or epochs, gradually reducing the network's error on the training data.

### 3.4 Optimizer (ADAM)

Optimization algorithms are crucial for training neural networks by adjusting parameters (weights and biases) to minimize the loss function. While basic Stochastic Gradient Descent (SGD) updates parameters based on the gradient of the loss for a single training example or a mini-batch, more advanced optimizers aim for faster convergence and better performance.

ADAM, which stands for Adaptive Moment Estimation, is one such popular and effective optimization algorithm. It combines the advantages of two other SGD extensions: AdaGrad, which adapts learning rates per-parameter, and RMSProp, which also uses per-parameter adaptive learning rates based on a moving average of squared gradients. ADAM computes adaptive learning rates for each parameter by storing an exponentially decaying average of past squared gradients (like RMSProp) and an exponentially decaying average of past gradients (like momentum).

Let $L(\boldsymbol{\theta})$ be the loss function with parameters $\boldsymbol{\theta}$, and $g_t = \nabla_{\boldsymbol{\theta}} L_t(\boldsymbol{\theta}_{t-1})$ be the gradient at timestep $t$. ADAM maintains two moving averages: First moment vector (mean of gradients):

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t \tag{11}$$

Second moment vector (uncentered variance of gradients):

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2 \tag{12}$$

where $\beta_1$ and $\beta_2$ are hyperparameters, typically close to 1 (e.g., 0.9 and 0.999 respectively). Since $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$ are initialized as 0-vectors, they are biased towards zero, especially during the initial timesteps. ADAM corrects for this bias:

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1-\beta_1^t} \tag{13}$$

$$\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1-\beta_2^t} \tag{14}$$

The parameter update rule is then:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon} \tag{15}$$

where $\alpha$ is the learning rate (step size) and $\epsilon$ is a small constant (e.g., $10^{-8}$) to prevent division by zero. ADAM is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data and/or parameters.

### 3.5 Loss Functions

An error function, also known as a loss function or cost function, quantifies the discrepancy between the predicted output of a model and the actual target values. The choice of error function is critical as it guides the learning process of the neural network. For most regression and time series tasks, Mean Squared Error (MSE) and Mean Absolute Error (MSE) are the two most common choices.

MSE is one of the most widely used loss functions for regression. It is defined as the average of the squared differences between predicted values ($\hat{y}_i$) and actual values ($y_i$):

$$L_{\text{MSE}}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{16}$$

where $N$ is the number of samples. MSE penalizes larger errors more significantly due to the squaring operation. This makes it sensitive to outliers, as a few large errors can dominate the total loss and excessively influence the model's parameters. However, MSE is a smooth, differentiable function, which makes it amenable to gradient-based optimization.

**Huber Loss** (Huber, 1964) offers a compromise between MSE and Mean Absolute Error (MAE). It behaves like MSE for small errors and like MAE for large errors, making it less sensitive to outliers than MSE. The Huber loss is defined by a parameter $\delta$, which acts as a threshold:

$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{for } |y - \hat{y}| \leq \delta \\ \delta\left(|y - \hat{y}| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \tag{17}$$

For small errors ($|y - \hat{y}| \leq \delta$), Huber loss is quadratic, providing smooth gradients. For large errors ($|y - \hat{y}| > \delta$), it is linear, which not only reduces the penalty for outliers compared to MSE, but also provides a more stable gradient, when the model guesses wrong, which is ideal when forecasting prices where we want to avoid the local minima of predicting Repeat. The parameter $\delta$ controls the transition point. Huber loss is differentiable everywhere and combines the benefits of strong penalization of small errors (like MSE) with robustness to outliers (like MAE). Financial time series often contain unexpected large movements (outliers), making Huber loss a potentially more robust choice than MSE for such data.

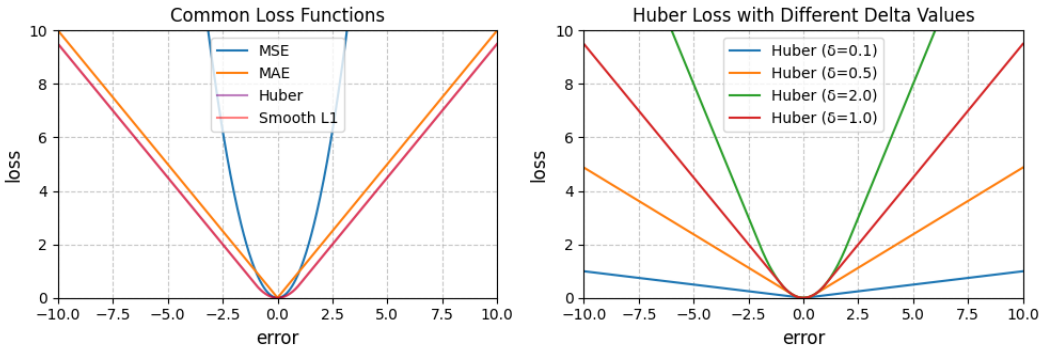



Figure 2: Loss functions comparison

### 3.6 Recurrent Neural Network (RNN)

RNNs (Elman, 1990) are a class of neural networks specifically designed to process sequential data, such as time series, natural language, or speech. Unlike feedforward neural networks, RNNs possess internal memory through recurrent connections, allowing them to capture temporal dependencies and context from previous inputs in the sequence.

The core idea of an RNN is to use the same set of weights and biases for each element in the sequence. At each time step $t$, an RNN takes an input $x_t$ and its hidden state $h_{t-1}$ from the previous time step to compute the new hidden state $h_t$. This hidden state acts as a summary of the information seen so far in the sequence. The computation can be expressed as:

$$\boldsymbol{h}_t = f_h(W_{xh}\boldsymbol{x}_t + W_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h) \tag{18}$$

where $\boldsymbol{x}_t$ is the input at time $t$, $\boldsymbol{h}_{t-1}$ is the hidden state from the previous time step, $W_{xh}$ and $W_{hh}$ are weight matrices for the input and recurrent connections respectively, $\boldsymbol{b}_h$ is a bias vector, and $f_h$ is an activation function (often tanh or ReLU). The output $\boldsymbol{y}_t$ at time step $t$ can then be computed from the hidden state:

$$\boldsymbol{y}_t = f_y(W_{hy}\boldsymbol{h}_t + \boldsymbol{b}_y) \tag{19}$$

where $W_{hy}$ is a weight matrix and $\boldsymbol{b}_y$ is a bias vector, and $f_y$ is an output activation function.

RNNs are trained using a variation of backpropagation called Backpropagation Through Time (BPTT). This involves "unrolling" the network in time, creating a deep feedforward network where each layer corresponds to a time step, and then applying standard backpropagation. While powerful, simple RNNs suffer from the vanishing or exploding gradient problem. This makes it difficult for them to learn long-range dependencies in sequences, as gradients propagated over many time steps tend to either shrink to zero or grow excessively large, hindering effective learning.

### 3.7 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) networks are a specialized type of RNN architecture designed to overcome the vanishing and exploding gradient problems, enabling them to effectively learn long-range dependencies in sequential data. LSTMs achieve this through a more complex internal structure involving a memory cell and gating mechanisms.

An LSTM unit consists of: **Cell State ($\boldsymbol{C}_t$):** This acts as the network's "memory." Information can be added to or removed from the cell state, regulated by gates. It runs relatively unchanged through the entire chain of LSTM units, allowing information to persist over long sequences. **Gates:** These are neural network layers (typically with sigmoid activation) that control the flow of information into and out of the cell state and the hidden state.

**Forget Gate ($\boldsymbol{f}_t$)**: Decides what information to discard from the previous cell state $\boldsymbol{C}_{t-1}$.

**Input Gate ($\boldsymbol{i}_t$)**: Decides which new information to store in the current cell state $\boldsymbol{C}_t$. It works in conjunction with a $\tanh$ layer that creates a vector of new candidate values, $\tilde{\boldsymbol{C}}_t$.

**Output Gate ($\boldsymbol{o}_t$)**: Decides what part of the cell state $\boldsymbol{C}_t$ to output as the hidden state $\boldsymbol{h}_t$.

The key equations for an LSTM unit at time step $t$ are (where $\sigma$ is the sigmoid function and $\odot$ denotes element-wise multiplication):

$$\boldsymbol{f}_t = \sigma(W_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f) \tag{20}$$

$$\boldsymbol{i}_t = \sigma(W_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i) \tag{21}$$

$$\tilde{\boldsymbol{C}}_t = \tanh(W_C[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_C) \tag{22}$$

$$\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{C}}_t \tag{23}$$

$$\boldsymbol{o}_t = \sigma(W_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o) \tag{24}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t) \tag{25}$$

Here, $W_f, W_i, W_C, W_o$ are weight matrices and $\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_C, \boldsymbol{b}_o$ are bias vectors for the respective gates and cell computations. $[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]$ denotes the concatenation of the previous hidden state and the current input.

The gating mechanisms allow LSTMs to selectively remember or forget information over extended periods, making them highly effective for tasks involving complex temporal patterns, such as machine translation, speech recognition, and time series forecasting.

### 3.8 Fourier Transformation

The Fourier Transform is a fundamental mathematical tool that decomposes a signal (a function of time or space) into its constituent frequency components. It provides a way to represent a signal in the frequency domain, where the characteristics of the signal are described in terms of amplitudes and phases of different sinusoidal waves.

For a continuous-time signal $x(t)$, the Continuous Fourier Transform (CFT) $X(f)$ is defined as:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi ft} dt \tag{26}$$

where $f$ is the frequency, $t$ is time, and $i = \sqrt{-1}$ is the imaginary unit. The inverse transform reconstructs the time-domain signal from its frequency-domain representation:

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{i2\pi ft} df \tag{27}$$

In practice, signals are often discrete, sampled at regular intervals. For a discrete signal $x[n]$ of length $N$, the Discrete Fourier Transform (DFT) is used:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i2\pi kn/N} \quad \text{for } k = 0, 1, \ldots, N-1 \tag{28}$$

Here, $X[k]$ represents the complex amplitude of the $k$-th frequency component. The DFT essentially projects the signal onto a basis of $N$ complex sinusoids with frequencies $f_k = k/N\Delta t$, where $\Delta t$ is the sampling interval. The Fourier Transform is invaluable in signal processing for tasks such as spectral analysis (identifying dominant frequencies), filtering (removing or attenuating specific frequency bands), and compression.

### 3.8.1 Fast Fourier Transform (FFT)

An FFT is not a new transform but rather an efficient algorithm for computing the Discrete Fourier Transform (DFT) and its inverse. A direct computation of the DFT for a sequence of length $N$ requires $O(N^2)$ complex multiplications and additions. For large $N$, this complexity can be prohibitive. The discrete Fourier transform (DFT) is given by the formula:

The FFT significantly reduces this computational burden to $O(N \log N)$ operations. This dramatic speed-up is achieved by exploiting symmetries and periodicities in the DFT definition, recursively breaking down the DFT of size $N$ into smaller DFTs. The most common FFT algorithms are the Cooley-Tukey algorithm (which works well when $N$ is a power of 2, but can be adapted for other $N$) and the prime-factor algorithm.

The FFT significantly reduces this computational burden to $O(N \log N)$ operations. This speed-up is primarily achieved by algorithms like the Cooley-Tukey method, which exploits symmetries and periodicities inherent in the DFT. The core idea is to recursively break down a DFT of size $N$ into smaller DFTs. For instance, if $N$ is a power of two (a common case for which these algorithms are highly optimized), a DFT of size $N$ can be efficiently computed from two DFTs of size $N/2$ — one for the even-indexed inputs ($E_k = \text{DFT}\{x_{2m}\}$) and one for the odd-indexed inputs ($O_k = \text{DFT}\{x_{2m+1}\}$). The full DFT components $X_k$ are then reconstructed using the "butterfly" operations:

$$X_k = E_k + W_N^k O_k \tag{29}$$

$$X_{k+N/2} = E_k - W_N^k O_k \tag{30}$$

for $k = 0, \ldots, N/2 - 1$, where $W_N^k = e^{-i2\pi k/N}$ is a complex root of unity known as a "twiddle factor". This recursive "divide and conquer" strategy, when applied $\log_2 N$ times, is the source of the $O(N \log N)$ complexity. While the Cooley-Tukey algorithm is most straightforward for $N$ being a power of 2, variations exist for arbitrary $N$.

The number of components in the frequency domain is given by:

$$\frac{N}{2} + 1 \quad \text{if } N \text{ is even } \frac{N+1}{2} \quad \text{if } N \text{ is odd}$$

## 3.9 Smoothing Techniques and Why

Smoothing techniques are widely employed in time series analysis to mitigate noise and reveal underlying patterns or trends. Raw time series data, particularly financial data, often exhibit random fluctuations or high-frequency oscillations that can obscure the signal of interest and complicate modeling efforts. The primary objective of smoothing can be summed up as noise reduction. By filtering out these seemingly unpredictable (random), short-term variations not considered part of the fundamental underlying process, we potentially improve the signal-to-noise ratio (SNR). For

predictive tasks, this not only help models from overfitting to spurious noise, it also helps to highlight longer-term trends or cyclical patterns with thet aim of leading to better learning and generalization.

Several smoothing techniques exist, each with distinct characteristics and mathematical formulations, and can be grouped as filters in two distinct categories: causal and non-causal. **Moving Averages (MA)** are the most commonly known filters and are causal filters.

The **Simple Moving Average (SMA)** computes the unweighted mean of the previous $k$ data points $P_t, P_{t-1}, \ldots, P_{t-k+1}$. For a price $P_t$ at time $t$, the SMA $S_t$ is given by:

$$S_t = \frac{1}{k} \sum_{i=0}^{k-1} P_{t-i} \tag{31}$$

The **Exponential Moving Average (EMA)** assigns exponentially decreasing weights to older observations, making it more responsive to recent price changes than an SMA of similar length. The EMA $E_t$ is calculated recursively:

$$E_t = \alpha P_t + (1 - \alpha) E_{t-1} \tag{32}$$

where $P_t$ is the current price, $E_{t-1}$ is the previous EMA value, and $\alpha$ is the smoothing factor, often related to a period $N$ by $\alpha = 2/(N + 1)$. MAs are straightforward to implement but inherently introduce lag. The choice of the window size $k$ for SMA or the smoothing factor $\alpha$ for EMA is crucial and can be subjective, significantly impacting the trade-off between smoothness and responsiveness.

**Gaussian Filters** is also a causal filter and involves convolving the time series $x[n]$ with a Gaussian kernel $g[n]$. This provides a weighted average where points closer to the center of the kernel (the current point being smoothed) receive higher weights. The Gaussian kernel is defined as:

$$g[n] = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{n^2}{2\sigma^2}} \tag{33}$$

where $n$ is the offset from the center of the kernel, and $\sigma$ is the standard deviation, which controls the degree of smoothing. The smoothed signal $y[t]$ is then the result of the convolution:

$$y[t] = (x * g)[t] = \sum_{j=-M}^{M} x[t - j] \cdot g[j] \tag{34}$$

for a symmetric kernel of length $2M + 1$, often normalized so that $\sum g[j] = 1$.

A critical consideration in selecting smoothing techniques for financial forecasting is their impact on signal timeliness and feature preservation for deep learning models. While Simple Moving Averages (SMA) and Exponential Moving Averages (EMA) are common, they introduce phase lag where the smoothed signal trails actual price developments. Though lag can be corrected by shifting the series, this requires future data and is unsuitable for real-time forecasting. Causal moving averages using only past data therefore reflect delayed signals. Additionally, averaging can blur sharp price movements and turning points, potentially discarding valuable signal content.

### 3.10 Motivation for FFT smoothing

From the perspective of signal processing, it is a common observation across various domains that noise often manifests predominantly in the higher frequency components of a signal, while the underlying signal of interest resides in the lower frequencies (Oppenheim & Schafer, 1975). By attenuating or removing high-frequency components with a **Low-pass Filter**, it allows low-frequency components to pass. If this assumption holds, then the signal of interest primarily resides in lower frequencies (the most repeating patterns), while noise is concentrated in higher frequencies. In financial markets, high-frequency fluctuations can arise from market microstructure effects, algorithmic trading, or other short-term phenomena. If a significant portion of this high-frequency content constitutes noise, attenuating these components could yield a cleaner signal for modeling. This could also explain why we often see deep learning models struggle to outperform the expectation of a random walk (Repeat baseline) especially on financial datasets, as highlighted by studies such as Zeng et al. (2022).

So if we accept that price data inherently contains some amount of noise, then neural networks, in their attempt to approximate the data in its entirety, will backpropagate errors from noisy data, and obviously struggle model sequences to meaningful patterns. This leads to gradient instability, the model learning an error-minimizing baseline (e.g., predicting a repeat of the last value), and/or overfitting to noise rather than the underlying signal. Noise, by its stochastic nature, represents fundamentally unpredictable fluctuations rather than learnable, deterministic patterns. Consequently, a hypothesis guiding this work is that optimally smoothing the input series could enhance the generalization ability of these deep learning architectures.

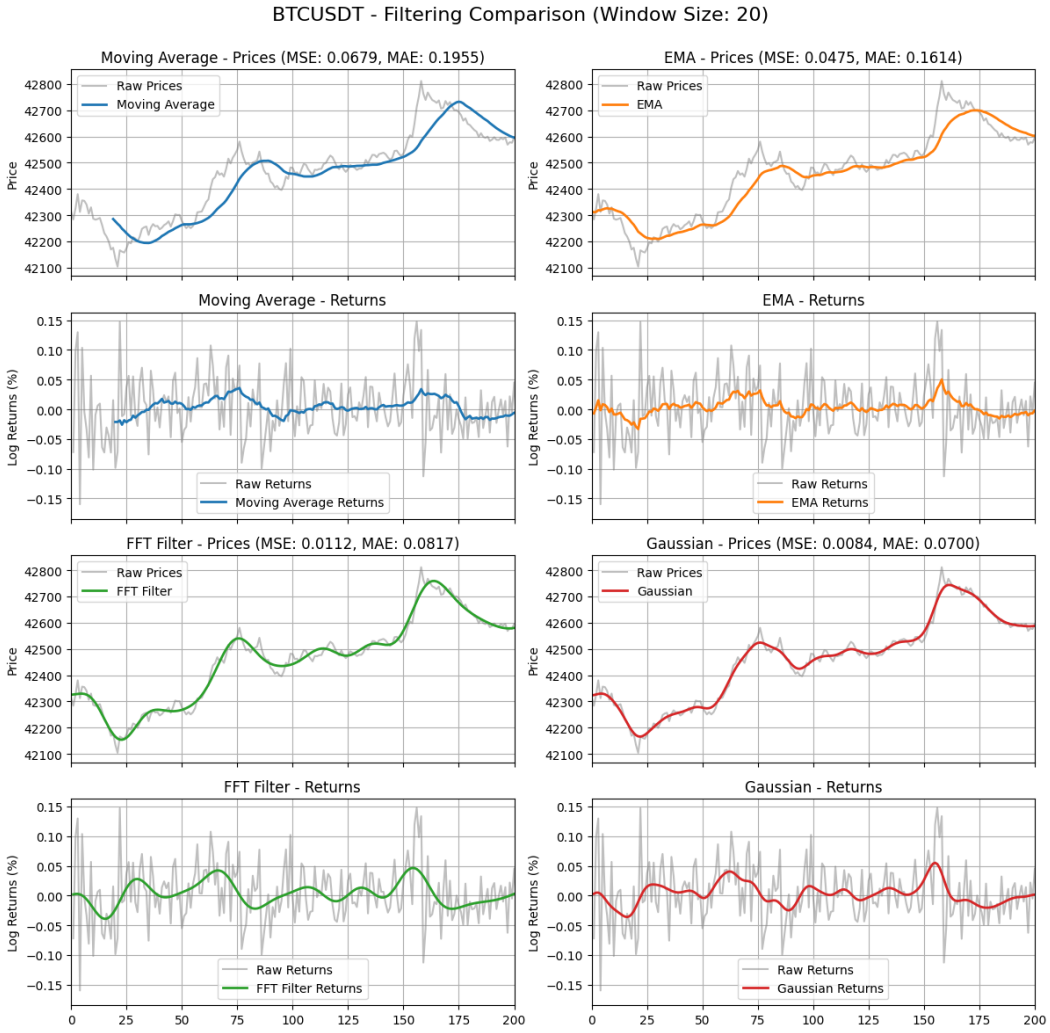



Figure 3: Comparison of EMA, SMA, FFT and Gaussian filters

Figure 3 presents a visual comparison of SMA, EMA, FFT, and Gaussian filters applied to BTCUSDT price data, their corresponding log returns, and error scores calculated on standardized prices. Notably, time-domain filters like Gaussian filters can achieve a close fit to the raw price data, with the Gaussian filter in this example yielding the lowest MSE of 0.0084 and MAE of 0.0700 against the raw prices, slightly outperforming the FFT filter (MSE of 0.0112, MAE of 0.0817).

However, when considering the input for a predictive model like an LSTM, particularly when operating on transformed data such as log returns, the characteristics of the smoothed series itself become paramount. Observing the log returns plots in Figure 3 (bottom row), the FFT-smoothed log returns visually appear to exhibit a more pronounced sinusoidal or periodic character compared to those smoothed by the Gaussian filter. This also makes sense given the nature of how Fourier Transforms

work, i.e. decomposing a time series into its patterns of sine and cosine waves by how often they repeat.

It is therefore hypothesized that this enhanced regularity, periodicity, and more defined wave-like structure in the FFT-smoothed log returns might be more amenable to learning by an LSTM, which excels at capturing recurrent patterns and temporal dependencies.

While the Gaussian filter provides a marginally closer approximation to the raw price series in terms of simple error metrics, the decision to focus on FFT-based smoothing for this thesis is driven by this hypothesis. The enhanced responsiveness and phase accuracy of FFT filtering at the most recent edge of the data segment, combined with this characteristic in particularly in the log returns domain, are posited to provide a more beneficial input for the predictive model. Future work could involve a more exhaustive comparative analysis of how different filtering methods, including Gaussian and others, quantitatively impact the predictive performance of the LSTM across various datasets and forecast horizons.


(a) Candlesticks with SMA

(b) Candlesticks with FFT

Figure 4: Comparison of log returns from a SMA and FFT filter to highlight smoothness

In essence, the goal is to identify an optimal cutoff frequency for the FFT filter that preserves the maximum amount of predictive information while filtering out the maximum amount of noise.

### 3.11 Risk Management

A famous quote by Warren Buffett, widely considered the one of the best investors of all time, is:

"Rule No. 1: Never lose money. Rule No. 2: Never forget Rule No. 1."

This emphasis stems from the inherent asymmetry of percentage returns; a substantial loss requires a disproportionately larger gain to recover the initial capital. For instance, as illustrated in Figure 5, a 50% portfolio decline necessitates a 100% gain to break even, while a 95% loss demands a staggering 1900% (or 19-fold) return. Clearly, preempting such drawdowns is more pragmatic than attempting to recover from them.

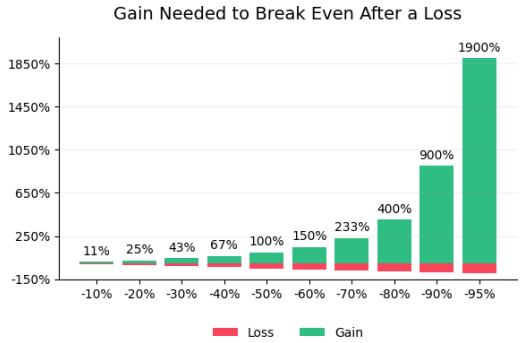



Figure 5: Asymmetry of percentage gains and losses

A widely adopted metric, which evaluates the risk-adjusted return of an investment or strategy, is the Sharpe ratio. It is defined as:

$$S_a = \frac{\mathbb{E}[R_a - R_b]}{\sigma_a} \tag{35}$$

where $\mathbb{E}[R_a - R_b]$ represents the expected excess return of the asset or strategy (with return $R_a$) over a benchmark risk-free rate (with return $R_b$), and $\sigma_a$ is the standard deviation of these excess returns, quantifying volatility. A higher Sharpe ratio generally indicates a more favorable return for the amount of risk undertaken. While the Sharpe ratio considers total volatility, variations like the Sortino ratio offer a nuanced perspective by focusing specifically on downside volatility, aligning more directly with the goal of minimizing painful drawdowns. However, the Sharpe ratio remains the most common standard for performance comparison.

The three ways to mitigate risk are: 1. Diversify, 2. Make fewer, more accurate bets. 3. Scale your bets to your edge or confidence.

While these ratios are valuable for retroactive evaluation and a fast comparison, a more crucial aspect of risk management is determining the appropriate allocation of capital to specific strategies or assets proactively. The Kelly criterion seeks to maximize long-term capital growth by systematically relating bet size to perceived advantage, or statistical edge.

### 3.12 A New Interpretation of Information Rate (The Kelly Criterion): Optimal Bet Sizing

In 1956, John Kelly Jr. made a profound connection between Shannon's information theory and optimal betting strategy. Kelly's insight was that a gambler receiving information through a noisy communication channel faces a problem mathematically equivalent to maximizing information transmission rate.

Consider a gambler with access to a channel that transmits information about the outcomes of events before they become public knowledge. This scenario maps directly to a communication channel where the actual outcome represents the input $X$, the signal received by the gambler represents the output $Y$, and transmission errors occur with probability $p$. Shannon had shown that the capacity of such a binary symmetric channel is:

$$C = 1 + p \log_2 p + (1-p) \log_2 (1-p) \tag{36}$$

In modern notation, this equals the mutual information $C = \max_{P(X)} I(X;Y) = \max_{P(X)} [H(Y) - H(Y|X)]$, though Shannon originally wrote this as an explicit sum rather than using the compact notation $I(X;Y)$ that became standard later.

Kelly considered a gambler who bets a fraction $f$ of their wealth on each favorable signal. For a binary outcome with probability $p$ of winning and odds $b:1$, the wealth after $N$ bets becomes

$$V_N = V_0 \cdot (1+bf)^W \cdot (1-f)^L \tag{37}$$

where $W$ and $L$ are the number of wins and losses. The exponential growth rate of wealth is therefore

$$G(f) = \lim_{N\to\infty} \frac{1}{N} \log_2 \frac{V_N}{V_0} = p \log_2(1 + bf) + (1 - p) \log_2(1 - f) \tag{38}$$

To find the optimal betting fraction, we maximize $G(f)$ by setting its derivative to zero:

$$\frac{dG}{df} = \frac{pb}{1 + bf} - \frac{1 - p}{1 - f} = 0 \tag{39}$$

Solving yields the Kelly criterion:

$$f^* = \frac{pb - (1 - p)}{b} = \frac{p(b + 1) - 1}{b} \tag{40}$$

For even-money bets where $b = 1$, this simplifies to $f^* = 2p - 1$. Kelly's remarkable discovery was that substituting this optimal fraction back into the growth rate formula gives $G(f^*) = C$, exactly Shannon's channel capacity.

In the general case with multiple outcomes and arbitrary odds $\alpha_s$, Kelly showed that the growth rate is

$$G = \sum_{s,r} p(s, r) \log_2 \alpha_s a(s/r) \tag{41}$$

where $a(s/r)$ is the fraction bet on outcome $s$ given signal $r$. When odds are "fair" meaning $\alpha_s = 1/p(s)$, the maximum growth rate becomes

$$G_{\max} = H(X) - H(X|Y) = I(X; Y) \tag{42}$$

precisely Shannon's mutual information between the source and received signal.

The Kelly criterion has since been extended to modern portfolio theory. Consider an investor allocating fraction $f$ of wealth to a risky asset with expected return $\mu$ and variance $\sigma^2$, keeping fraction $(1 - f)$ in a risk-free asset with return $r$. The portfolio return is $R_p = (1 - f)r + fR$ where $R$ is the asset's random return. For small time intervals, the expected logarithmic growth rate is approximately

$$G(f) \approx E[\log(1 + R_p)] \approx E[R_p] - \frac{1}{2}\text{Var}(R_p) \tag{43}$$

Since $E[R_p] = r + f(\mu - r)$ and $\text{Var}(R_p) = f^2\sigma^2$, we have

$$G(f) \approx r + f(\mu - r) - \frac{1}{2} f^2\sigma^2 \tag{44}$$

Maximizing with respect to $f$ gives

$$\frac{dG}{df} = (\mu - r) - f\sigma^2 = 0 \tag{45}$$

Therefore, the optimal allocation is

$$f^* = \frac{\mu - r}{\sigma^2} \tag{46}$$

This is the continuous-time Kelly criterion, showing that the optimal fraction is the excess return divided by variance, like the the Sharpe ratio.

The profound insight is that optimal capital allocation is fundamentally about efficiently encoding one's information advantage into position sizes, with the growth rate limited by the quality of information—exactly as Shannon's theory predicts for communication channels. The Kelly criterion tells us that the optimal betting fraction is proportional to the edge (information advantage), the maximum growth rate equals the mutual information between private signals and outcomes, and overbetting reduces long-term growth despite short-term possibilities. This connection between information theory and optimal betting remains one of the most elegant applications of Shannon's work outside traditional communication systems.

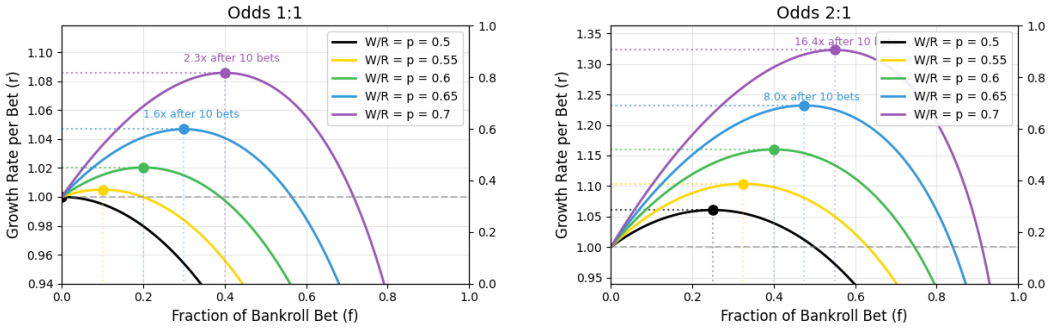



Figure 6: Kelly Criterion Growth Rates

### 3.13 Maximizing Win Rate (Condorcet's Jury Theorem)

If your statistical edge, or win rate, directly corresponds to the amount of money you make, then maximizing your win rate will maximize your returns. But how can we improve our win rate if we have already squeezed out all the edge of the model itself? Intuitively, it seems that the win rate of the global portfolio will be the same as the win rate of the local portfolios. However, thinking it through, assuming equal Profit and Loss (PnL) on average, if 100 different independent bets are made at the same time each with 55% chance of winning, then you would expect 55 portfolios to be in profit on average at any given time. But what is the chance that at least 51 of them are, i.e. the global majority? Well, that would be:

$$P_{\text{majority}}(N, p) = \sum_{k=\lceil N/2 \rceil}^{N} \binom{N}{k} p^k (1-p)^{N-k} \tag{47}$$

What we have just rediscovered here is Condorcet (1785)'s jury theorem but for portfolio management. The theorem states that if each individual has an independent probability $p > 0.5$ of making a correct judgment on a binary issue, then as the number of individuals increases, the probability that the majority of the group will make the right decision approaches 1 as group size $N$ tends to infinity.

Likewise, if we assume a global portfolio consisting of local uncorrelated (independent) portfolios, then the chance that the global portfolio's PnL will be positive at a given timestep (majority of the local portfolios are in profit) also follows this law, assuming equal volatility.

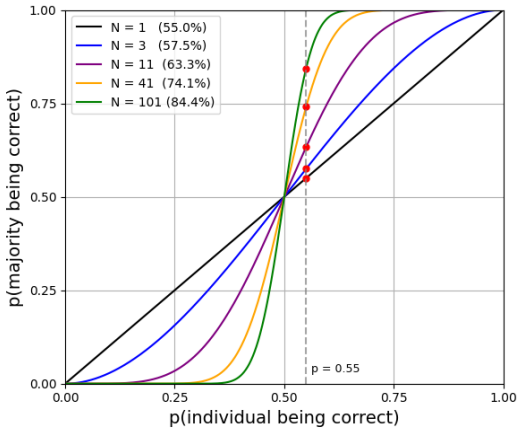



Figure 7: Condorcet's jury theorem

So on top of already minimizing our drawdown and volatility by diversifying our portfolio with uncorrelated strategies, we ALSO increase the global portfolio's average win rate, allowing us to compound our returns at an even faster rate.

### 3.14 Portfolio Variance Theory

As we know from before, in portfolio theory, the objective is maximizing risk-adjusted returns. This is most often done through diversification of **uncorrelated** assets, but Portfolio Variance Theory by Markowitz (1952), and extended by Fernholz (2002), counterintuitively find that the best risk-adjusted returns can actually be found by combining **inversely** correlated assets with different weights to "cancel out" the variance.

Consider a portfolio of two assets with weights $w_1$ and $w_2$ and variances $\sigma_1^2$ and $\sigma_2^2$ and correlation $\rho_{12}$. Then they show that the variance of the combined portfolio $\sigma_p^2$, becomes:

$$\sigma_p^2 = \sum_{i=1}^{N}\sum_{j=1}^{N} w_i w_j \sigma_i \sigma_j \rho_{ij} = w_1^2\sigma_1^2 + w_2^2\sigma_2^2 + 2w_1 w_2 \sigma_1 \sigma_2 \rho_{12} \tag{48}$$

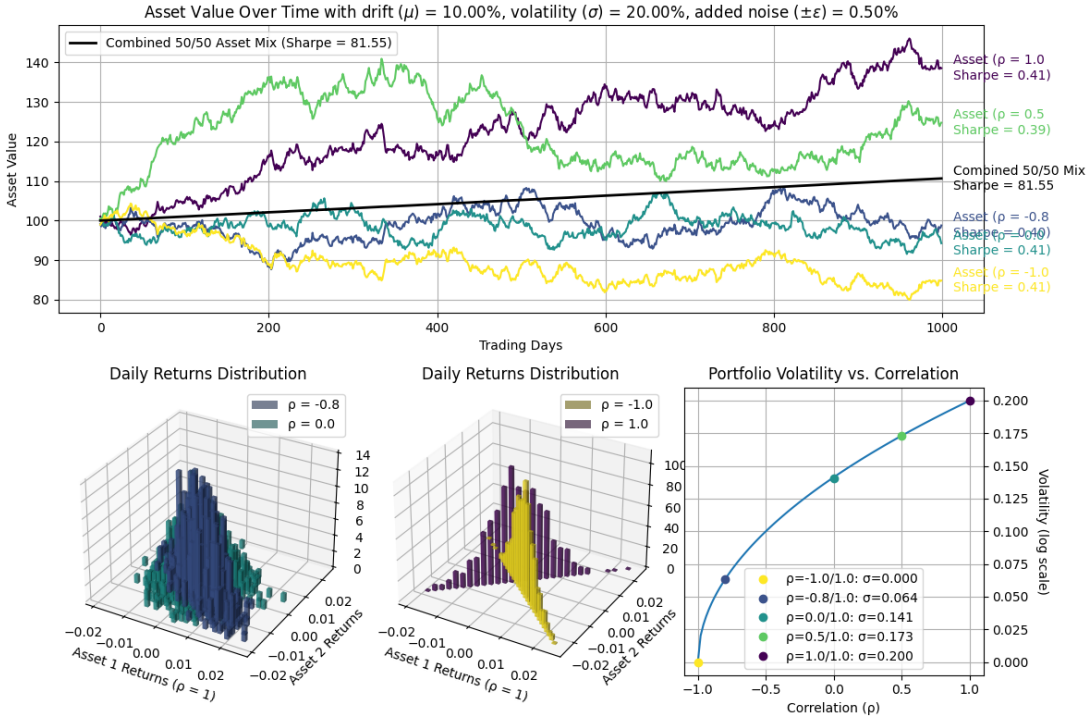
For example, for a portfolio with 10% returns and 20% volatility and equal weights, their variance becomes:

$$\sigma_p^2 = (0.5)^2 \cdot (0.2)^2 + (0.5)^2 \cdot (0.2)^2 + 2 \cdot (0.5) \cdot (0.5) \cdot (0.2) \cdot (0.2) \cdot \rho_{12}$$
$$\sigma_p^2 = 0.25 \cdot 0.04 + 0.25 \cdot 0.04 + 0.5 \cdot 0.04 \cdot \rho_{12}$$
$$\sigma_p^2 = 0.02 + 0.02 \cdot \rho_{12}$$

When $\rho_{12} = 0$, $\sigma_p^2 = 0.02$, so $\sigma_p = \sqrt{0.02} \approx 0.141$ or $14.1\%$.
When $\rho_{12} = -1$, $\sigma_p^2 = 0.02 - 0.02 = 0$, thus $\sigma_p = 0\%$.

When the correlation coefficient is zero ($\rho_{12} = 0$), the cross-terms vanish, leaving us with a smaller risk profile than indivudally. However, with perfect negative correlation ($\rho_{12} = -1$), these terms become negative, actively reducing portfolio variance.

If we visualize this, what we find is actually very interesting. As we would expect, the variance of two combined portfolio (purple line) is effectively canceled out and just becomes a straight line to the top (which is what we want).




**Note: Noise was added to prevent exploding Sharpe ratios.*

Figure 8: Portfolio variance theory, $\rho$ = correlation coefficient

However, when we visualize it in 3D, it becomes very clear why. The yellow line shows two perfectly correlated assets, and as we can see, the variance is actually just amplified. And although we end up with slightly higher returns, the risk profile associated leaves us with a Sharpe ratio of 0.40. Compared to the inversely correlated assets (purple line), which with a combination of 50/50 weights, ends up with a Sharpe ratio of 82.48. It actually ended up at $10^{12}$ originally without noise, but this is a common quirk of sharpe ratios (whenever the denominator is close to 0, the sharpe ratio explodes:):

# 4 Data

To avoid recomputing the same redundant FFT during the forward pass of the model, especially when dealing with longer datasets, we pre-process the entire time series by applying FFT smoothing once to the entire time series, and the filtered time series is then used as input to the model.

## 4.1 Common Benchmarking Datasets

| Dataset | ETTh1 | ETTh2 | ETTm1 | ETTm2 | Weather | Exchange |
|---|---|---|---|---|---|---|
| Prediction ($y$) | OT | OT | OT | Channels | Last col. | Last col. |
| Description of $y$ | Oil Temp. | Oil Temp. | Oil Temp. | Oil Temp. | kWh | ?/USD |
| Sampling Rate | 1 hour | 1 hour | 15 min | 15 min | 10 min | 1 day |
| Start date | 2016-07-01 | 2016-07-01 | 2016-07-01 | 2016-07-01 | 2020-01-01 | 1990-01-01 |
| End date | 2018-06-26 | 2018-06-26 | 2018-06-26 | 2018-06-26 | 2021-01-01 | 2010-10-10 |
| Channels (columns) $\times$ | 7 | 7 | 7 | 7 | 21 | 7 |
| Timesteps (rows) | 17,420 | 17,420 | 69,680 | 69,680 | 52,696 | 7,588 |
| Size (MB) | 2.6 | 2.4 | 10.4 | 9.7 | 10.4 | 0.6 |

In order to ensure consistency between evaluation of DLinear, Time-MoE, and previous time series model papers, the train, validation, and test data splits are specified very precisely according these splits:

| Dataset | ETTh1 | ETTh2 | ETTm1 | ETTm2 | Weather | Exchange |
|---|---|---|---|---|---|---|
| Train Timesteps | 8640 | 8640 | 34560 | 34560 | 36887 | 5311 |
| Val Timesteps | 2880 | 2880 | 11520 | 11520 | 5270 | 760 |
| Test Timesteps | 2880 | 2880 | 11520 | 11520 | 10539 | 1517 |

It also turns out ETTh1 and ETTm1 are the same dataset as ETTh2 and ETTm2, but sampled at different rates. So ETTm1 and ETTm2 are just more granular versions of ETTh1 and ETTh2.

## 4.2 Binance Cryptocurrency Data

20 GB of open high low close volume (OHCLV) csv data was fetched from `https://data.binance.vision/` through the python library binance-historical-data to serve as the financial datasets for this thesis. It contains data for all cryptocurrency symbols listed on Binance from 2022-01-01 to today with 1 minute granularity.

As we learned previously, the optimal variance-adjusted portfolio is the one where inversely correlated assets are combined with weights that minimize the total portfolio variance. Of those, the 33 symbols found to be the most inversely correlated are shown in the correlation matrix (Figure 9), however, after filtering out delisted symbols (leading to repeating values and 0 std), and symbols with less than 10,000 data points date, we are left with 28 symbols.

| Dataset | BTCUSDT | | | {CHILLGUY, ..., 1000000MOG}USDT |
|---|---|---|---|---|
| Prediction ($y$) | Close | | | Close |
| Description of $y$ | Close | | | Close |
| Start date | 2025-01-01 | | | 2022-01-01 (earliest) |
| End date | 2025-04-27 | | | 2025-04-27 (common) |
| Channels (columns) $\times$ | 5 | | | 5 |
| Timesteps (rows) | 167,679 | 55,913 | 335,48 | 17,420 |
| Sampling Rate | 1m | 3m | 5m | 1m |
| Size (MB) | 17 | 3.5 | 4.1 | 172.4 |

The data has been verified programmatically to match exactly with data fetched directly from the real-time Binance Futures API. It doesn't contain all symbols (more recent ones are missing), and

some symbols are still in the dataset despite being delisted (e.g. BUSD), meaning they contain repeating values after the delist date. These are all filtered out before analyzed for correlation.

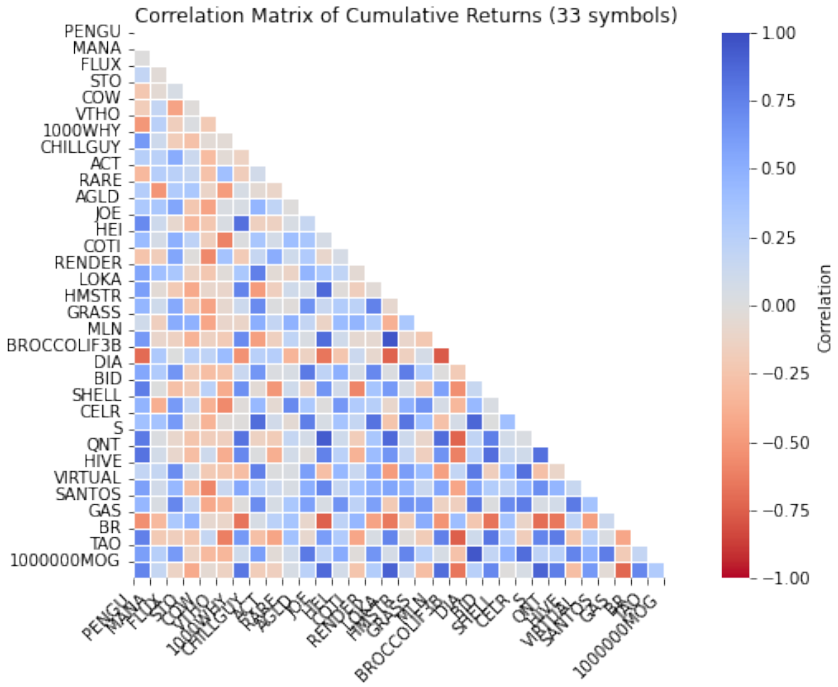



Figure 9: Correlation Matrix of symbols (pre filtering)

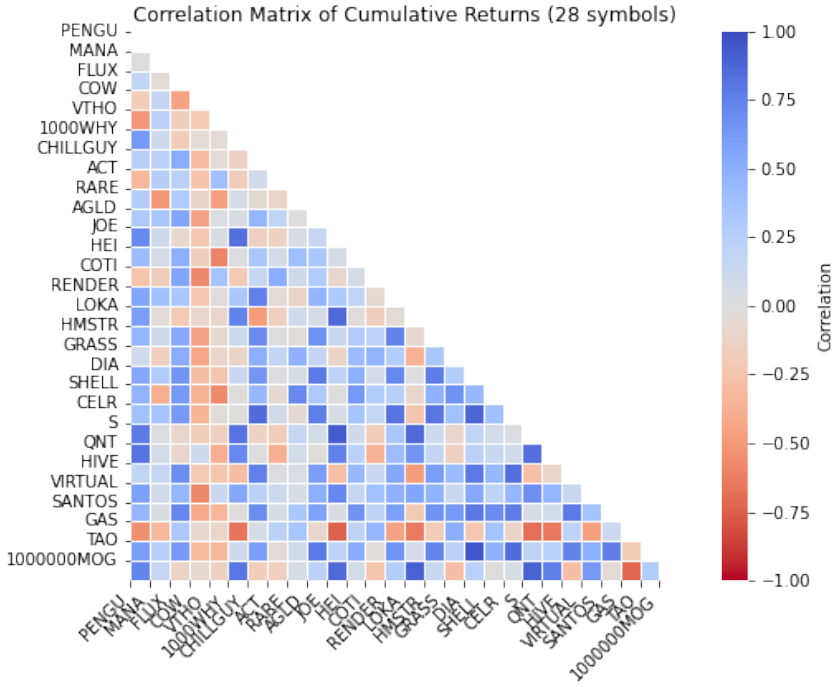



Figure 10: Correlation Matrix of symbols (post filtering)

# 5 Methods

## 5.1 FFT Smoothing

To isolate the presumed underlying signal from high-frequency noise, we employ a filtering technique based on the Fast Fourier Transform (FFT). The FFT provides an efficient means of decomposing a time series from the time domain into the frequency domain, representing the signal as a sum of sinusoids of different frequencies, amplitudes, and phases. By applying a low-pass filter in the frequency domain, we can filter out high-frequency noise and preserve lower frequency (higher amplitude) components that we hypothesize contain more predictable and arbitrageable patterns.

The process follows these steps and can be formalized as:

$$X(\omega) = \mathcal{F}[x(t)] \quad \text{(FFT)}$$
$$X_{\text{filtered}}(\omega) = X(\omega) \cdot H(\omega) \quad \text{(Low Pass Filter)}$$
$$x_{\text{filtered}}(t) = \mathcal{F}^{-1}[X_{\text{filtered}}(\omega)] \quad \text{(IFFT)}$$

Where $H(\omega)$ is the frequency response of the low-pass filter:

$$H(\omega) = \begin{cases} 1, & \text{if } |\omega| \leq \omega_c \\ 0, & \text{if } |\omega| > \omega_c \end{cases}$$

With $\omega_c$ representing the cutoff frequency.

The result is a smoothed version of the original time series, where rapid fluctuations have been removed. The cutoff frequency dictates the aggressiveness of the smoothing.

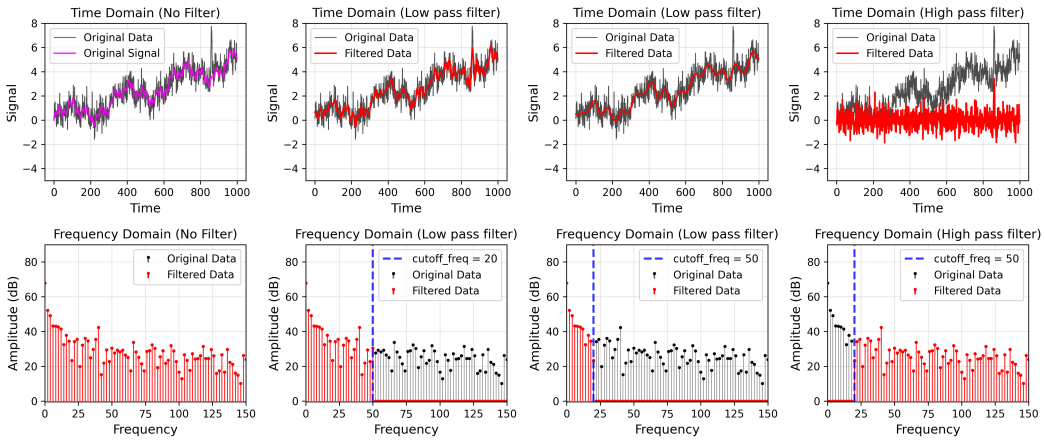



Figure 11: FFT Smoothing

As suggested by information-theoretic principles (i.e., Shannon's channel capacity concept, more to come), there exists a trade-off: while filtering removes noise, excessive filtering (a very low cutoff frequency) can also remove valuable information contained in the higher frequencies, potentially degrading predictive performance

The cutoff frequency is chosen by window_size, representing the shortest cycle period we wish to retain in the smoothed signal. This parameter requires careful tuning, typically guided by empirical validation or domain knowledge about the characteristic time scales of market movements relevant to the forecasting horizon.

### 5.1.1 Ensuring Consistent Smoothing Across Variable Lengths

In practice, the length of a time series we choose to model can vary considerably. A crucial aspect of any filtering technique is the consistency of its smoothing effect, regardless of the input signal's length. With time-domain filters like moving averages, the window size (e.g., MA(20)) provides an easy-to-understand, intuitive, and consistent measure of smoothing aggressiveness, irrespective of whether the total signal is 100 or 1000 samples long.

For our frequency-based smoothing, the parameter analogous to a moving average's window size is the cutoff period, derived from the cutoff frequency $\omega_c$. In our implementation, we define the cutoff based on a parameter such that:

$$\omega_c = \frac{2\pi}{\text{min\_period\_samples} \cdot \Delta t} \quad \text{or} \quad f_c = \frac{1}{\text{min\_period\_samples} \cdot \Delta t}$$

where $\Delta t$ is the sampling interval (e.g., 1 minute). If we consider a normalized frequency (cycles per sample or radians per sample), where the sampling interval is implicitly unity ($\Delta t = 1$), the cutoff frequency $f_c^{\text{norm}}$ (in cycles/sample) becomes:

$$f_c^{\text{norm}} = \frac{1}{\text{min\_period\_samples}} = \frac{1}{\text{window\_size}}$$

This definition ensures that the filter removes frequency components corresponding to oscillations with periods shorter than the window size. For example, if the window size is set to 20, all cyclical components that complete a full cycle in fewer than 20 sampling intervals are attenuated.

This approach directly links the smoothing level to a specific time scale inherent in the data, defined by the number of samples. Provided the sampling rate ($\Delta t^{-1}$) is constant across different time series segments, which it is with price data, using a fixed window size will consistently filter out the same range of **physical** frequencies (e.g., cycles per 1m, 3m, 1h, etc.). This offers a consistent interpretation of smoothing aggressiveness, akin to a moving average window: it targets features based on their duration in terms of sampling intervals, not as a fraction of the total signal length. While the **absolute number** of high-frequency components removed might vary with signal length (longer signals potentially containing more such components), the **characteristic timescale** of the filtered-out variations remains constant.

Alternative approaches exist, but can lead to inconsistent smoothing levels. For instance, retaining the top N frequency components might result in a much lower effective cutoff frequency (and thus more aggressive smoothing) for a shorter, less complex signal compared to a longer, richer one. Our method ensures that the "feel" of the smoothing—the scale of details being removed remains consistent.

### 5.1.2 Mitigating Edge Artifacts

A well-known challenge when applying frequency-based filtering to finite-length time series segments is the emergence of artifacts at the edges of the signal when reverting back to the time domain after applying a filter. This is known as Gibbs phenomenon, and it happens because an implicit assumption of a Fourier Transform (FT) is that the signal is periodic, meaning $x(t + T) = x(t)$ for all $t$. For real-world, non-stationary time series like price data, this property is rarely satisfied, so a discontinuity exists at the boundaries where the signal $x(0) \neq x(T)$, aka not periodic. This discontinuity manifests as a jump function which, according to Fourier theory, introduces high-frequency components proportional to $1/f$ across the entire spectrum.[4]

When forecasting financial time series, these edge effects are particularly problematic as the most recent data points (at the right edge) often contain the most important information we need.

To mitigate this phenomenon, we employ padding before performing the FFT. Padding extends the time series segment artificially, effectively moving the original segment's endpoints away from the boundaries of the signal being transformed. Various padding methods exist, but each comes with their own set of drawbacks. For instance:

[4] You can read more about it here: Gibbs phenomenon, Wikipedia

- Zero padding: Extending the signal with zeros (creates artificial discontinuities)
- Mean padding: Extending with the signal mean (introduces bias)
- Reflection padding: Mirroring the signal at its boundaries to various lengths

If we think about it: The optimal approach should minimize the discontinuity that exists when the signal is viewed periodically. For a signal $x(t)$ defined on $[0, T]$, a complete reflection of itself creates an extended signal $x_r(t)$ where:

$$\begin{aligned} x_r(t) &= x(t) && \text{for } t \in [0, T] \\ x_r(t) &= x(2T - t) && \text{for } t \in [T, 2T] \end{aligned}$$

This extension guarantees that $x_r(0) = x_r(T)$, eliminating the boundary discontinuity when the signal is treated as periodic with period $2T$. Furthermore, this creates a signal with even symmetry around $t = T$:

$$x_r(T + \tau) = x_r(T - \tau) \quad \text{for all } \tau \in [0, T] \tag{49}$$

According to Fourier theory, signals with even symmetry have purely real Fourier coefficients, eliminating phase distortions in the frequency domain. Additionally, the complete reflection ensures that any frequency component in the original signal completes an integer number of half-cycles in the extended signal, minimizing spectral leakage.

Through empirical evaluation (see Appendix/Code Section X demonstrating benchmark_padding), padding the entire signal with a 1:1 reflected version of itself (ratio = 1.0) achieved the lowest edge effect and MSE from filtered return xy. This mathematical reasoning is validated by the experimental results, confirming that complete reflection creates the most natural periodic extension for FT processing of financial time series.

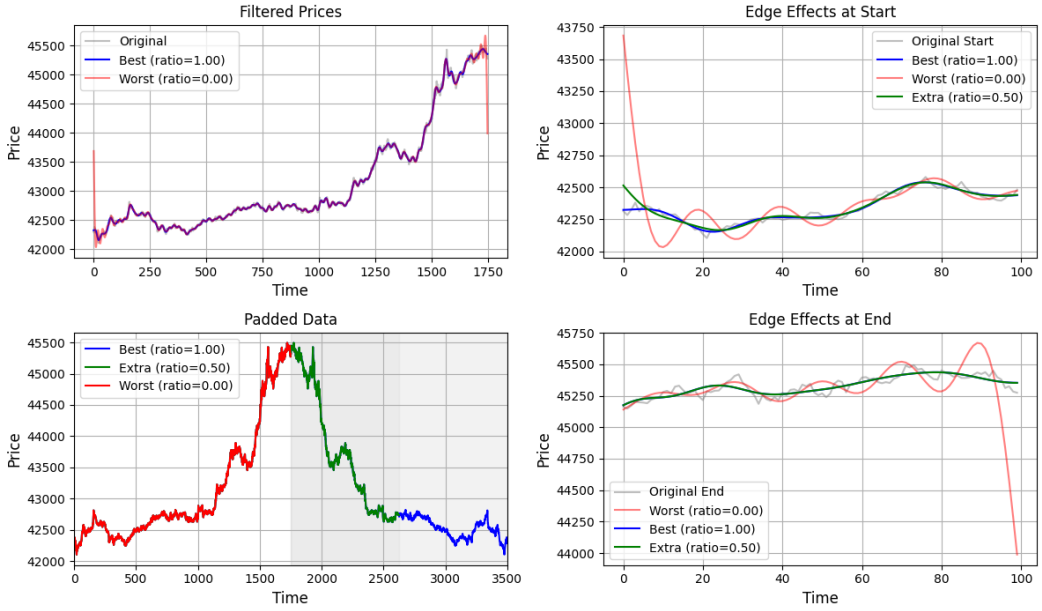



Figure 12: Padding FFT-Smoothed Data with Comparison (window size = 20)

### 5.2 Differentiation, Log Returns, and Stationarity

We transform the price data into log returns to achieve stationarity. Converting prices to returns results in a stationary series with mean equal to its drift / trend (typically 0) and taking the logarithm of those returns symmetrizes the distribution of returns such that their range is $(-\infty, \infty)$, unlike

simple returns which are bounded by $(-100\%, \infty)$. So taking the log returns of a time series effectively detrends the signal, making every new time step independent of the previous price as opposed to moving with the previous price, which enables the models to focus solely on capturing relevant price movements without losing any information.

When inspecting the forecasts of earlier models, it becomes clear that they most often over- or undershoot raw prices, failing to capture the underlying trend. In the log returns domain, the same models also often learn an error-minizing strategy which involves predicting something close to 0 (no change, repeat).

However, while examining the smoothed prices, it occurred that the log returns of the **fft smoothed prices** would appear much more stable, sinusoidal, and periodic, given that the jittery high-frequency components (or noise) have been removed.

Specifically, log returns are calculated using the formula at every time step:

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right) \times 100, \tag{50}$$

where $P_t$ denotes the price at time $t$. The inverse transformation from log returns back to prices is:

$$P_t = P_{t-1} \times e^{\frac{r_t}{100}}. \tag{51}$$

### 5.3 Model Architecture

Sequence-to-sequence LSTM networks are particularly renowned for their ability to capture the sequential dependencies, hence why it was used. The LSTM's gating mechanisms allow it to selectively retain or discard information over extended sequences, addressing the vanishing gradient problem that plagues standard recurrent neural networks when modeling long-range dependencies.

Our model ingests a sequence of length seq_len of smoothed log returns as input features. The core computational structure comprises two stacked LSTM layers that progressively extract and refine temporal patterns from the input data. The first LSTM layer processes the univariate input sequence, transforming each time step's scalar value into a 64-dimensional hidden representation. Formally, for each time step $t$ in the input sequence, the first LSTM computes:

$$\mathbf{h}_t^{(1)} = \text{LSTM}_1(\mathbf{x}_t, \mathbf{h}_{t-1}^{(1)}, \mathbf{c}_{t-1}^{(1)}) \tag{52}$$

where $\mathbf{x}_t$ represents the input at time $t$ (the smoothed log return), and $\mathbf{h}_{t-1}^{(1)}$ and $\mathbf{c}_{t-1}^{(1)}$ are the previous hidden state and cell state, respectively, each with dimension 64. This first layer processes the entire sequence, generating a sequence of hidden states $\{\mathbf{h}_1^{(1)}, \mathbf{h}_2^{(1)}, ..., \mathbf{h}_{\text{seq\_len}}^{(1)}\}$ that encodes progressively higher-level temporal features.

Unlike the first layer where we utilize the full output sequence, we extract only the final hidden state $\mathbf{h}_{\text{seq\_len}}^{(2)}$ from the second LSTM layer. This 64-dimensional vector serves as a compressed contextual representation of the entire input sequence, encapsulating the relevant temporal patterns deemed most informative for forecasting future values. This design choice follows the intuition that the final hidden state contains the accumulated knowledge processed through the entire sequence, with the recurrent nature of LSTMs ensuring that information from earlier time steps is appropriately propagated forward with selective retention of salient patterns.

The extracted hidden state is then passed through a fully connected linear layer that projects this 64-dimensional representation to a vector of length seq_len + pred_len:

$$\mathbf{y} = W\mathbf{h}_{\text{seq\_len}}^{(2)} + \mathbf{b} \tag{53}$$

where $W \in \mathbb{R}^{(seq\_len+pred\_len)\times 64}$ and $\mathbf{b} \in \mathbb{R}^{seq\_len+pred\_len}$ are the weight matrix and bias vector of the dense layer, respectively. The output vector $\mathbf{y}$ contains predicted values for both the input sequence (the first seq_len elements) and the future time steps (the last pred_len elements).

For our forecasting application, we isolate the last pred_len elements of $\mathbf{y}$, which constitute the model's prediction for future log returns:

$$\hat{\mathbf{r}}_{t+1:t+pred\_len} = \mathbf{y}_{seq\_len+1:seq\_len+pred\_len} \tag{54}$$

This output is then reshaped to match the expected target tensor dimensions by adding a feature dimension, resulting in a tensor of shape $[batch\_size, pred\_len, 1]$.

The architecture's design reflects several deliberate considerations. The two-layer LSTM configuration provides a hierarchical extraction of temporal features, with the first layer identifying basic patterns in the smoothed log returns, and the second layer capturing higher-order dependencies between these patterns. The choice of 64 hidden units strikes a balance between model capacity and computational efficiency, providing sufficient representational power while mitigating the risk of overfitting, particularly important given the noisy nature of financial data even after FFT smoothing.

Furthermore, the direct mapping from the final hidden state to both historical and future time steps through the dense layer leverages the model's learned representation to simultaneously reconstruct the input sequence and extend it into the future. While we only utilize the future predictions, this joint reconstruction-prediction objective can potentially enhance the model's ability to capture the underlying temporal dynamics of the time series.

During training, the model parameters are optimized to minimize a predefined loss function, typically Huber loss as discussed earlier, which provides robustness against outliers while maintaining sensitivity to small errors. The model's gradients are computed through backpropagation through time (BPTT) and parameters are updated using the ADAM optimizer, which adaptively adjusts learning rates for each parameter based on gradient history.

This LSTM architecture, coupled with the FFT smoothing preprocessing and log return transformation discussed in previous sections, forms a comprehensive framework for financial time series forecasting that addresses the challenges of noise, non-stationarity, and complex temporal dependencies inherent in market data.

$$\mathbf{h}_t^{(2)} = \text{LSTM}_2(\mathbf{h}_t^{(1)}, \mathbf{h}_{t-1}^{(2)}, \mathbf{c}_{t-1}^{(2)}) \tag{55}$$

Unlike the first layer where we utilize the full output sequence, we extract only the final hidden state $\mathbf{h}_{\text{seq\_len}}^{(2)}$ from the second LSTM layer. This 64-dimensional vector serves as a compressed contextual representation of the entire

### 5.4 Searching for Uncorrelated / Inversely Correlated Assets

In order to maximize the global probability of being correct (according to Condorcet's Jury Theorem), we want to include as many independent symbols as possible, which means minimizing the amount of pairwise correlation between symbols.

Several algorithms were developed to select a sub-set, or basket, of assets from the processed pool, guided primarily by their pairwise correlation characteristics.

One method employed a stochastic optimization technique, simulated annealing, to identify a basket of a specified size. This process commenced with a randomly selected initial basket of symbols. Iteratively, new candidate baskets were proposed by randomly swapping one symbol from the current best basket with another symbol from the available pool. The quality of each basket was evaluated using a scoring mechanism. This internal scoring function offered three distinct strategies based on the chosen optimization type. An 'inverse' strategy scored baskets based on the count of pairwise correlations below a given threshold, thus favoring baskets with more negatively correlated or very low correlation pairs. An 'uncorrelated' strategy used a score equivalent to the negative mean of the absolute pairwise correlations (i.e., $-$mean(|correlation values|)), directly incentivizing the selection of assets whose correlations were as close to zero as possible. A 'hybrid' strategy utilized a custom score that positively weighted strongly negative correlations (e.g., those less than $-0.7$) and near-zero correlations (e.g., those between $-0.2$ and $0.2$), while penalizing positive correlations (e.g., those greater than $0.3$). A candidate basket was accepted if its score surpassed that of the current best. If the score was worse, the candidate could still be accepted with a probability dependent

on the score difference and a gradually decreasing "temperature" parameter, a feature allowing the search algorithm to escape local optima. This optimization process continued for a predetermined number of iterations.

Another approach involved a greedy selection process to construct an asset basket, with options for subsequent refinement. The initial construction phase began with a single symbol, iteratively adding new symbols to the basket. In each step, the algorithm selected the candidate symbol that, when added to the current basket, yielded the best improvement in the basket's score. The scoring method during this initial greedy construction varied. For an 'original' method, and during the greedy phases of an Iterated Local Search (ILS) refinement, the score was calculated as the negative sum of all positive pairwise correlations within the basket (formally, $-\sum \max(\text{pairwise correlation in sub-basket}, 0)$). This score is maximized when all pairwise correlations are zero or negative. Alternatively, if an 'annealing' based method was specified for greedy construction, the 'hybrid' scoring strategy, as described previously, was used. Optional refinement stages could follow. If Iterated Local Search was employed, the initial greedy basket was perturbed by removing some symbols and then re-filling the basket greedily for a specified number of iterations. If an 'annealing' refinement method was chosen, the initial greedy basket was further refined using a simulated annealing process, similar to the dedicated optimization described earlier, for a set number of iterations, again using the 'hybrid' score.

A third, specialized greedy algorithm was designed to directly target assets with correlations close to zero. This method also started with a single symbol and iteratively added symbols to the basket until the desired basket size was reached. In each step, it evaluated all candidate symbols not yet included in the basket. The score for adding a candidate symbol was determined by the negative mean of the absolute pairwise correlations of the temporary basket that included the candidate (i.e., $-\text{mean}(|\text{correlations of temporary basket}|)$). The candidate yielding the highest score—which corresponds to the lowest mean absolute correlation, indicating correlations closest to zero—was added to the basket. This method directly implemented a strategy focused on identifying assets with pairwise correlation values near zero.

### 5.5 Combining the Optimal Weights for Minimizing Variance

Once an asset basket was selected through any of the aforementioned strategies, a subsequent procedure was employed to determine the allocation weights that would minimize the portfolio's variance. This procedure utilized the previously generated price data and the list of selected symbols as input. Logarithmic returns were first calculated for the selected symbols from their price series. Following this, the covariance matrix, denoted $\mathbf{\Sigma}$, of these logarithmic returns was computed. A numerical optimization was then performed, typically using a Sequential Least Squares Programming (SLSQP) method. The objective function for this optimization was the minimization of the portfolio variance, mathematically expressed as $\mathbf{w}^T\mathbf{\Sigma}\mathbf{w}$, where $\mathbf{w}$ represents the vector of asset weights.

The optimization was subject to two primary constraints: first, the sum of all weights $\mathbf{w}$ must equal one (i.e., $\sum w_i = 1$), ensuring full investment of capital. Given that we want to maximize the amount of times we compound our returns, i.e. close out positions, we don't actually care about finding the best risk-adjusted return, but rather just reducing the variance as much as possible (because both LONG and SHORT signals can be traded). Figure 13 shows the results of this optimization.

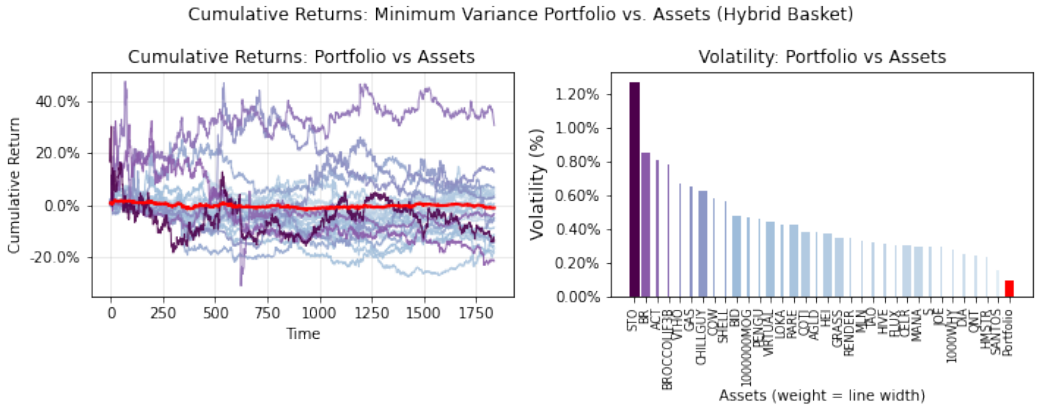



Figure 13: Portfolio Optimization

Caveat: this technique does not incorporate closing out positions, instead just holding over the entire period, but it is still a better proxy for the optimal portfolio than combining correlated assets.

### 5.6 Training setup

The model is trained using the AdamW optimizer and a custom Huber loss function (or standard Huber loss, depending on configuration) to predict these future pred_len smoothed log returns. The specific hyperparameters, such as seq_len (input sequence length, e.g., 992) and pred_len (prediction horizon, e.g., 32), are detailed in our experimental setup (Section X.X or Appendix Y.Y).

Learning rate scheduler CosineAnnealingLR.

# 6 Results

We tested this, then that. Then that. Different window sizes comparing the loss vs raw loss. Actually, it doesn't make sense to look at the loss of raw returns because the returns of the smoothed signal are much more stable. But raw normalized prices.

## 6.1 Methodology

It doesn't make senese to look at MSE RMSE RRMSE as in the end you are only realy comparing an error score vs the Repeat baseline, and given that they all are derived from the square error, the relative difference is the same. Therefore, MSE, MAE, and Huber.

## 6.2 Optimal window size

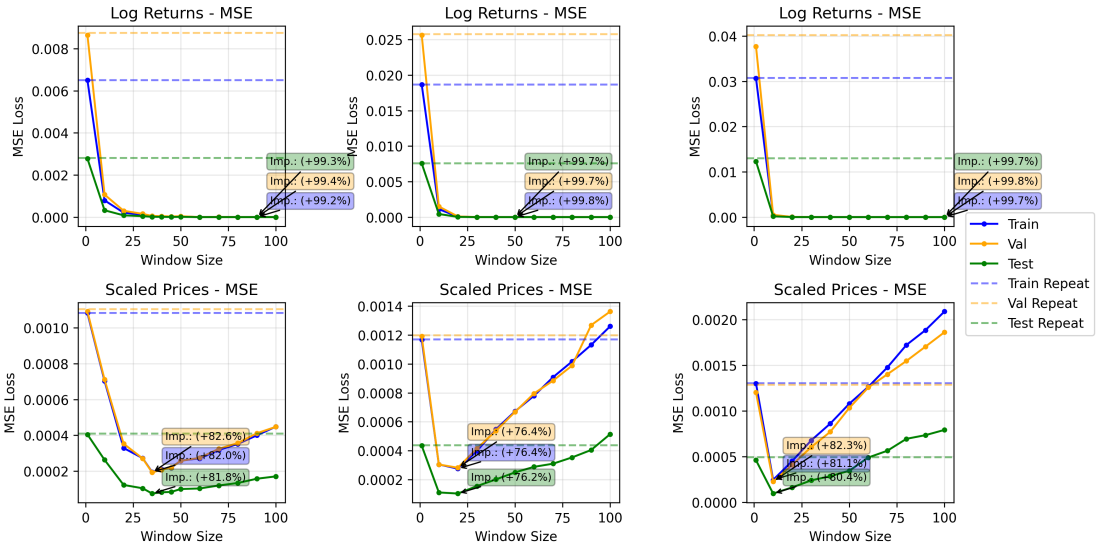



Figure 14: Window sizes for the BTC dataset on equivalent forecast horizons

In Figure 14, we present the results of an empirical study to determine the optimal FFT smoothing window size for Bitcoin (BTCUSDT) price data at different sampling granularities. To achieve this, a systematic hyperparameter sweep was conducted using our LSTM-based forecasting model. The core methodology involved training and evaluating the model across a predefined range of FFT window sizes: $\{1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$.

The experiment was performed on three variations of the BTCUSDT dataset, each with a different sampling frequency: 1-minute, 3-minute, and 5-minute intervals, all ending on 2025-01-01 for consistency in the data period analyzed. A crucial aspect of this study was to ensure comparability across these granularities. Therefore, the input sequence length (seq_len) and prediction horizon (pred_len) were adjusted for each dataset to represent approximately equivalent lookback periods and forecast horizons in absolute time. Specifically:

For the 1-minute dataset an input sequence length of 1000 data points (approx. 16.7 hours) was used to predict the subsequent 24 data points (24 minutes).

For the 3-minute dataset an input sequence length of 333 data points (approx. 16.65 hours) was used to predict the subsequent 8 data points (24 minutes).

For the 5-minute dataset an input sequence length of 200 data points (approx. 16.7 hours) was used to predict 5 data points (25 minutes) ahead.

All experiments utilized log returns as the input feature. The models were trained for 5 epochs with a batch size of 256, using the Adam optimizer and Huber loss. The performance metric used

to evaluate and compare the different window sizes was the Mean Squared Error (MSE) on the predicted log returns against the true future log returns, evaluated on train, validation, and test splits. The figure displays these MSE values, allowing for the identification of window sizes that yield the lowest prediction error for each data granularity.

From figure 14, we can see that the optimal window size is 30 for the 1m dataset, 20 for the 3m dataset, and 10 for the 5m dataset across train/val/test (meaning it generalizes.) Interestingly, this corresponds

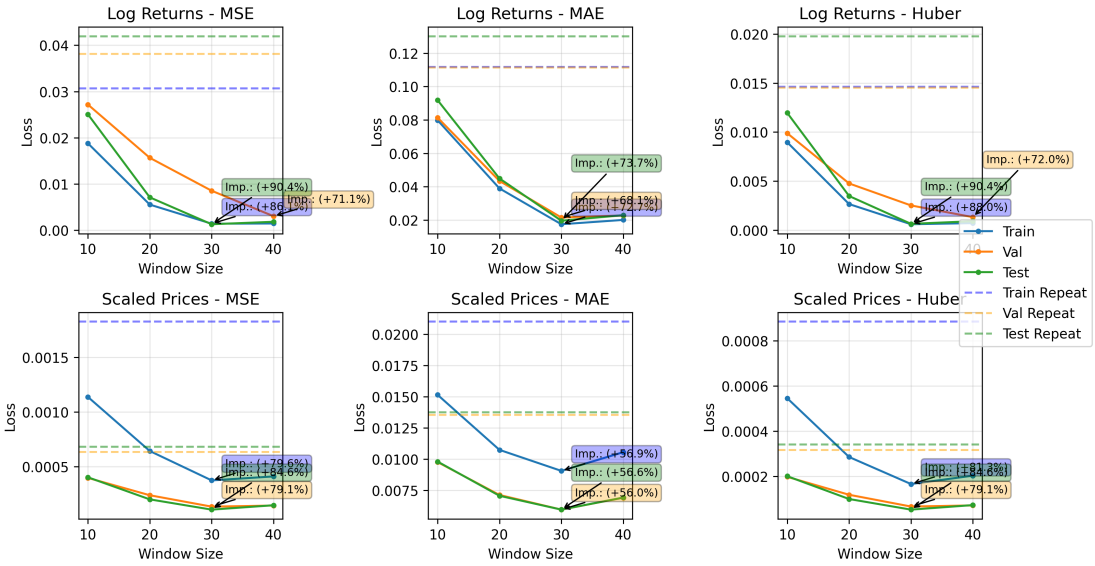



Figure 15: Window sizes for the 28 cryptos dataset

### 6.3 Benchmarking Datasets

| | | | FFTLSTM | | | DLinear | | TimeMoE-50M | | Repeat | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | PL | Imp. | WS | MSE | MAE | MSE | MAE | MSE | MAE | MSE | MAE |
| Weather | 96 | 1.4% | 10 | **0.001** | 0.025 | 0.005 | 0.056 | 0.003 | 0.043 | 0.001 | **0.025** |
| | 192 | 1.1% | 1 | **0.002** | 0.028 | 0.006 | 0.063 | 0.002 | 0.036 | 0.002 | **0.028** |
| | 336 | 3.3% | 1 | **0.002** | 0.031 | 0.006 | 0.067 | 0.002 | 0.035 | 0.002 | **0.031** |
| | 720 | -0.2% | 5 | 0.002 | 0.035 | 0.007 | 0.070 | 0.003 | 0.044 | **0.002** | **0.035** |
| ETTh1 | 96 | 52.6% | 90 | **0.027** | **0.124** | 0.057 | 0.179 | 0.072 | 0.197 | 0.069 | 0.203 |
| | 192 | 28.0% | 90 | **0.054** | **0.178** | 0.075 | 0.207 | 0.106 | 0.244 | 0.092 | 0.236 |
| | 336 | 24.9% | 90 | **0.073** | **0.208** | 0.097 | 0.243 | 0.177 | 0.329 | 0.113 | 0.265 |
| | 720 | 37.1% | 90 | **0.081** | **0.223** | 0.172 | 0.340 | 0.351 | 0.493 | 0.129 | 0.283 |
| ETTh2 | 96 | -12.8% | 5 | 0.123 | 0.266 | 0.132 | 0.280 | **0.109** | **0.253** | 0.295 | 0.423 |
| | 192 | -13.7% | 10 | 0.166 | 0.314 | 0.176 | 0.329 | **0.146** | **0.293** | 0.337 | 0.462 |
| | 336 | -29.9% | 1 | 0.229 | 0.377 | 0.210 | 0.369 | **0.176** | **0.328** | 0.390 | 0.502 |
| | 720 | 1.8% | 1 | **0.218** | **0.373** | 0.296 | 0.443 | 0.222 | 0.379 | 0.437 | 0.531 |
| ETTm1 | 96 | 62.1% | 80 | **0.011** | **0.076** | 0.029 | 0.125 | 0.045 | 0.156 | 0.033 | 0.136 |
| | 192 | 34.9% | 90 | **0.028** | **0.123** | 0.043 | 0.152 | 0.073 | 0.197 | 0.049 | 0.169 |
| | 336 | 29.6% | 90 | **0.045** | **0.160** | 0.064 | 0.187 | 0.098 | 0.228 | 0.065 | 0.196 |
| | 720 | 11.1% | 100 | **0.072** | **0.208** | 0.081 | 0.212 | 0.239 | 0.371 | 0.089 | 0.232 |
| ETTm2 | 96 | 34.4% | 80 | **0.042** | **0.151** | 0.064 | 0.185 | 0.101 | 0.237 | 0.228 | 0.354 |
| | 192 | 19.8% | 90 | **0.073** | **0.203** | 0.091 | 0.225 | 0.123 | 0.267 | 0.257 | 0.387 |
| | 336 | 9.8% | 90 | **0.111** | **0.252** | 0.123 | 0.265 | 0.171 | 0.316 | 0.287 | 0.414 |
| | 720 | 12.6% | 90 | **0.152** | **0.303** | 0.174 | 0.319 | 0.358 | 0.446 | 0.332 | 0.457 |
| Exchange | 96 | 78.4% | 150 | **0.019** | **0.107** | 0.099 | 0.242 | 0.123 | 0.277 | 0.088 | 0.221 |
| | 192 | 68.4% | 190 | **0.059** | **0.178** | 0.197 | 0.355 | 0.224 | 0.372 | 0.187 | 0.332 |
| | 336 | 31.4% | 190 | **0.218** | **0.341** | 0.318 | 0.458 | 0.358 | 0.486 | 0.370 | 0.468 |
| | 720 | -64.2% | 170 | 0.831 | **0.677** | 0.978 | 0.782 | **0.506** | 0.581 | 1.002 | 0.766 |

Table 1: **Univariate** forecasting errors on 'OT', the lower the better. The best results are highlighted in **bold** and the second best results are highlighted with underline. Imp. denotes the percentage improvement of FFTLSTM's MSE over the second best model's MSE (including Repeat). PL denotes the prediction length, i.e. the length of the forecast horizon.

| Dataset | | BTCUSDT | | | | | | 29 symbols | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| WS | Forecast | Imp. (LR) | Imp. (SP) | W/R | FF | FR | RR | Imp. (LR) | Imp. (SP) | W/R | FF | FR | RR |
| 25 | 1 | 99.2% | -19.0% | 96.7% | 97% | 57% | 58% | 99.6% | -25.2% | 98.4% | 98% | 57% | 51% |
| | 2 | 99.4% | 18.8% | 97.0% | 97% | 61% | 60% | 99.6% | 15.2% | 99.0% | 99% | 60% | 53% |
| | 5 | 97.1% | 56.7% | 94.6% | 96% | 69% | 65% | 99.3% | 57.4% | 98.5% | 99% | 68% | 59% |
| | 10 | 85.3% | 72.7% | 88.4% | 93% | 75% | 71% | 95.8% | 76.1% | 95.2% | 97% | 75% | 66% |
| | 30 | 33.0% | 37.9% | 61.5% | 76% | 70% | 68% | 59.7% | 70.7% | 72.8% | 90% | 80% | 74% |
| | 60 | 9.5% | 11.2% | 55.9% | 64% | 61% | 60% | 33.1% | 38.8% | 65.2% | 78% | 74% | 70% |
| | 120 | 1.4% | 4.9% | 53.4% | 54% | 53% | 53% | 20.2% | 19.5% | 58.3% | 69% | 66% | 64% |
| 30 | 1 | 99.6% | -40.1% | 98.2% | 98% | 56% | 58% | 99.8% | -49.5% | 99.5% | 99% | 56% | 50% |
| | 2 | 99.0% | 4.3% | 96.6% | 97% | 58% | 59% | 99.6% | -1.3% | 99.2% | 99% | 59% | 52% |
| | 5 | 96.0% | 48.9% | 93.4% | 95% | 66% | 63% | 99.3% | 49.2% | 99.3% | 99% | 66% | 58% |
| | 10 | 91.1% | 69.8% | 90.9% | 95% | 73% | 69% | 96.7% | 71.8% | 92.7% | 96% | 73% | 64% |
| | 30 | 34.2% | 40.0% | 60.4% | 77% | 70% | 68% | 67.0% | 76.1% | 54.3% | 63% | 59% | 56% |
| | 60 | 10.8% | 15.4% | 55.6% | 66% | 63% | 62% | 38.0% | 43.8% | 65.6% | 78% | 73% | 69% |
| | 120 | 1.8% | 5.4% | 53.1% | 55% | 54% | 54% | 20.4% | 24.0% | 60.5% | 69% | 67% | 64% |
| 35 | 1 | 99.6% | -65.9% | 97.9% | 98% | 55% | 57% | 99.7% | -72.6% | 99.4% | 99% | 55% | 49% |
| | 2 | 98.8% | -13.3% | 95.1% | 97% | 58% | 58% | 99.2% | -16.9% | 96.5% | 95% | 57% | 52% |
| | 5 | 97.8% | 39.9% | 95.2% | 97% | 65% | 63% | 98.4% | 41.3% | 98.3% | 99% | 64% | 57% |
| | 10 | 91.8% | 64.8% | 90.9% | 95% | 72% | 67% | 96.7% | 67.5% | 96.5% | 98% | 71% | 62% |
| | 30 | 45.5% | 54.2% | 66.1% | 82% | 73% | 70% | 70.1% | 80.7% | 78.2% | 90% | 78% | 71% |
| | 60 | 15.8% | 19.0% | 58.1% | 69% | 65% | 64% | 39.8% | 51.9% | 68.5% | 83% | 76% | 72% |
| | 120 | 3.0% | 7.1% | 53.4% | 56% | 55% | 55% | 22.4% | 28.3% | 59.8% | 71% | 68% | 65% |

Table 2: Univariate input on univariate columns. LR: Log Returns, SP: Scaled Prices. 29 symbols contains 26835 samples (test set). BTCUSDT_1m contains 16480 samples (test set).

### 6.4 Ablation Studies

| | Average MSE | Average MAE | Average Huber | Repeat |
|---|---|---|---|---|
| FFT-LSTM | 0.262 | 0.262 | 0.262 | 0.262 |
| window size 1 | 0.267 | 0.267 | 0.267 | 0.267 |
| window size 2 | 0.269 | 0.269 | 0.269 | 0.269 |
| window size 3 | 0.272 | 0.272 | 0.272 | 0.272 |
| window size 4 | 0.272 | 0.272 | 0.272 | 0.272 |
| window size 5 | 0.272 | 0.272 | 0.272 | 0.272 |
| window size 10 | 0.272 | 0.272 | 0.272 | 0.272 |
| window size 20 | 0.272 | 0.272 | 0.272 | 0.272 |

Table 3: On the dataset X in domain (price or log returns) after 10 epochs (best model by validation loss)

| | Average MSE | Average MAE | Average Huber | Repeat |
|---|---|---|---|---|
| FFT-LSTM | 0.262 | 0.262 | 0.262 | 0.262 |
| trained w Huber loss | 0.267 | 0.267 | 0.267 | 0.267 |
| trained w MAE loss | 0.269 | 0.269 | 0.269 | 0.269 |
| trained w MSE loss | 0.272 | 0.272 | 0.272 | 0.272 |
| trained w 32 hidden units | 0.272 | 0.272 | 0.272 | 0.272 |
| trained w 64 hidden units | 0.272 | 0.272 | 0.272 | 0.272 |
| trained w 128 hidden units | 0.272 | 0.272 | 0.272 | 0.272 |
| trained w/o shuffling training data | 0.275 | 0.275 | 0.275 | 0.275 |
| trained w/o auxiliary loss | 0.275 | 0.275 | 0.275 | 0.275 |

Table 4: On the dataset X in domain (price or log returns) after 10 epochs (best model by validation loss) with window size 10

# 7 Discussion

One thing you notice after training many of these models is that the longer the forecast horizon, the more the model tends to predict close to a flat line.

Given neural networks in time series apparently have a tendency to overfit to the noise / struggle to learn noisy data / look past the noise in data, the we want to find the optimal window_size that removes this noise while preserving as much of the signal as possible. However, in financial data, we don't know the underlying "signal" as we would with a transmission. So here's the dilemma: if we use mutual information between past and future as a measure, then the optimal signal becomes the signal that is most correlated with the future, i.e. removing everything to a flat line. If we use mutual information between the smoothed data and noisy data, then the window size becomes 1, because the noisy data is most representative of itself. So the only real way to find the optimal window size is to train a model on different window sizes and find the window size that minimizes the loss on the raw prices. Or what?

"Okay so it seems from your code that it's estimating the optimal based on a known true signal. I'm talking about the situations where you don't know the true underlying signal but have to guess how much noise to filter away quantitatively somehow. Something like this doesn't exist? You are absolutely right to push on this point. My previous code example assumed knowledge of the "true signal" S to demonstrate the RDT concept with a clear distortion target. In most real-world scenarios, especially with financial data, the true underlying "predictable signal" S is unknown. The short answer is: No, there isn't a universally agreed-upon, purely information-theoretic (like a direct RDT formula) method to quantitatively determine the optimal filter cutoff when the true underlying signal S is unknown, and your only observation is the noisy signal X. The problem shifts from "how to best represent S given X" to "how to best estimate S (or useful features of S) from X based on assumptions about S and the noise N." Why it's so hard when S is unknown: Undefined "Distortion" with respect to S: If S is unknown, you cannot directly calculate $D = E[(S - X_hat)^2]$. Any "optimality" criterion becomes dependent on your assumptions about S. Signal vs. Noise Ambiguity: What one considers "noise" to be filtered out, another might consider part of the complex "signal." This is especially true in finance where high-frequency fluctuations could be noise or could be short-term market dynamics. Task-Dependency: The "optimal" filter often depends on what you want to do with the filtered signal. A filter good for visual smoothing might not be good for input to a specific predictive model"

Log returns and fft smoothing effectively stationarizes / standardizes the data, similar to batch norm etc.

Double differenation to capture rate of change. Perhaps there is more signal in how quickly prices change?

perspektivering:

It is interesting how the big (scaling laws) transformer models seem to perform so bad from the DLinear paper, however, maybe with new smoothing techniques they will perform better.

Perhaps a model that includes some measure for when to execute trades (good scoops/deals) will be interesting.

Perhaps, most importantly of all, the reason it seems like these models perform so on time series tasks in the literature is because the benchmarking datasets are comically small. One big piece of evidence for this is that the loss for the train/val/test split are quite different, which means it's not fair to say they are sampled from the same distribution, or that the training data is not generalizable to the validation and test data.

This is also supported by the general observation that the bigger the dataset and the the more examples the models are trained on, the better they tend to perform.

Interestingly, searching for uncorrelated assets with the "hybrid basket" seemed to work better for volatility-minimized portfolio compared to searching for inversely correlated assets. This could be due to the fact that it's quite hard to find perfectly inversely correlated (-1) assets in the real world as opposed to uncorrelated (0) ones.

"3m contains more "signal" than 1m, 5m more than 3m, etc. Why?"

this could potentially be related to the Nyquist-Shannon Sampling Theorem. By sampling at 3m, 5m, even 1h as opposed to 1m, one could argue that

Connection to Nyquist-Shannon Sampling Theorem (Implicit): Theorem 13 on page 34 (”Let f(t) contain no frequencies over W...”) is the Nyquist-Shannon sampling theorem. It states that a signal perfectly band-limited to W Hz can be perfectly reconstructed from samples taken at a rate of 2W samples per second. This theorem underscores that the bandwidth W dictates the ”richness” or ”detail” of the signal. Reducing W means you are representing the signal with fewer independent pieces of information per unit time. In essence:

Shannon’s work formalizes the idea that bandwidth is a crucial resource for carrying information. While filtering can improve the quality (S/N ratio) of the information within a given band, if the act of filtering significantly reduces the total available bandwidth where the useful signal resides, it will inevitably limit the total amount of useful information that can be extracted or transmitted. Your statement correctly identifies this fundamental trade-off in the context of signal processing for prediction.

# 8 Conclusion

Answer research questions.

## References


L. Bachelier. Théorie de la spéculation. *Annales scientifiques de l'École Normale Supérieure*, 3e série, 17:21–86, 1900. doi: 10.24033/asens.476. URL https://www.numdam.org/articles/10.24033/asens.476/.

Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973. ISSN 00223808, 1537534X. URL http://www.jstor.org/stable/1831029.

Marie Jean Antoine Nicolas de Caritat Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. L'Imprimerie Royale, Paris, 1785.

Mark Davis and Alison Etheridge. *Louis Bachelier's Theory of Speculation: The Origins of Modern Finance (Translated from French)*. Princeton University Press, 2006. ISBN 9780691117522. URL http://www.jstor.org/stable/j.ctt7scn4.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Eugene F. Fama. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):383–417, 1970. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2325486.

Eugene F. Fama. Market efficiency, long-term returns, and behavioral finance. *Journal of Financial Economics*, 49(3):283–306, 1998.

Robert Fernholz. Stochastic portfolio theory. 01 2002. doi: 10.1007/978-1-4757-3699-1_1.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997. doi: 10.1162/neco.1997.9.8.1735.

Peter J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964. doi: 10.1214/aoms/1177703732.

Narasimhan Jegadeesh and Sheridan Titman. Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, 48(1):65–91, 1993.

Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–291, 1979.

Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2975974.

Maureen O'Hara. *Market Microstructure Theory*. Blackwell Publishing, 1995.

Alan V. Oppenheim and Ronald W. Schafer. *Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1975.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27 (3):379–423, 1948.

Robert J. Shiller. *Irrational Exuberance*. Princeton University Press, 2000.

Andrei Shleifer. *Inefficient Markets: An Introduction to Behavioral Finance*. Oxford University Press, 2000.

Richard H. Thaler. *Advances in Behavioral Finance*. Russell Sage Foundation, 1993.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting?, 2022. URL https://arxiv.org/abs/2205.13504.

# A Appendix

You may include other additional sections here.

## A.1 Benchmarking Padding Methods

## A.2 What was tried + failed + proof that high frequency noise is bad

NN learning just high frequency doesn't learn anything.