
# LSTM-Based Forecasting of Danish Electricity Imbalance Price

## A Comparative Study of Classical and Machine Learning Models

Christian Taulbjerg

**P10-Project, Matematik-Økonomi**

**Department of Mathematical Sciences**
Skjernvej 4A
DK-9220 Aalborg Ø
http://math.aau.dk

**AALBORG UNIVERSITET**
STUDENTERRAPPORT

**Title**
LSTM-Based Forecasting of Danish Electricity Imbalance Price

**Project period**
10. semester 2025

**Project group**
P10

**Authors**
Christian Taulbjerg

**Supervisors**
J. Eduardo Vera-Valdés

**Page number**
38

**Due date**
May 27, 2025

**Abstract**

Forecasting imbalance prices in electricity markets has become increasingly relevant due to volatility in part from renewable integration and regulatory reforms. This thesis explores the feasibility of modelling the Danish DK1 balancing market using LSTM, with the goal of enabling flexible assets to act after the closure of the ID market. A stateful LSTM model was trained under cuDNN constraints utilized GPU acceleration, mixed precision training, and memory optimization to support a hyperparameter tuning process that would otherwise have been computationally prohibitive. The tuning itself was performed using a combination of Hyperband and Bayesian Optimization. The LSTM model was benchmarked against ARIMA, naive baselines, and methods such as Random Forest and XGBoost. Results showed consistent improvements in predictive accuracy after transforming the target variable through co-integration (assuming unit root) with the day ahead spot price, capturing the structural anchoring relationship and improving the signal to noise ratio. Despite the complexity of LSTMs, simple AR models performed surprisingly well after transformation, underlining the short term dynamics that dominate the imbalance market. Ultimately, the optimized LSTM model demonstrated strong performance in capturing extreme price spikes and directional shifts, opening for post ID operations. The thesis highlight the importance of aligning model design with market structure and how simple models, under the right conditions can serve as an alternative.

*Rapportens indhold er frit tilgængeligt, men offentliggørelse (med kildeangivelse) må kun ske efter aftale med forfatterne.*

# Preface

This report is a 10th semester project in Mathematics-Economics at Aalborg University written in cooperation with Jysk Energi. The project title is *LSTM-Based Forecasting of Danish Electricity Imbalance Prices*. All programming was performed using Python or R, with environment management and GPU configuration handled through PowerShell and WSL2 (Windows Subsystem for Linux).

Generative AI tools have been used during the preparation of this project report as follows:

- **Coding Assistance**: Generative AI was utilized to accelerate coding tasks, particularly for routine and straightforward implementations, and to assist in debugging.
- **Language Improvement**: Generative AI was employed to correct grammatical errors, refine phrasing, and highlight areas where clarity was insufficient, improving the overall readability and precision of the report.

The use of generative AI was restricted to supporting tasks and did not replace critical analysis, decision-making, or the formulation of original ideas central to the project. All results, interpretations, and conclusions remain the product of my own work.

Christian Taulbjerg
Aalborg University, Maj 2025

# Table of Contents

# Figures

# 1 Introduction and Problem statement

The Danish and more broadly Nordic electricity balancing markets operate under a marginal pricing mechanism that has shown heightened volatility in recent years. This volatility reflects the growing introduction of renewable energy sources, physical limitations in grid infrastructure, and frequent regulatory adjustments. Collectively, these factors complicate forecasting efforts and pose challenges for maintaining system stability.

This thesis focuses on the analysis of hourly imbalance prices in the Danish electricity market, which are characterized by sharp spikes and heavy-tailed distributions. Such price behaviour is caused by a combination of structural and operational constraints: electricity demand remains largely inelastic in the short term, while the availability of flexible resources, such as gas turbines, is often limited during peak hours due to prior commitments or insufficient real-time responsiveness. Additionally, during periods of renewable overgeneration, reduction mechanisms are typically constrained, leaving the system with limited flexibility to manage real-time imbalances. These conditions frequently result in extreme and difficult-to-predict price movements.

Further complexity arises from structural shifts in the market, including the deployment of new cross-border interconnectors, frequent changes in reserve policy, and Denmark's participation in European platforms such as NBM, ENTSO-E, and PICASSO. These developments introduce nonlinear and often opaque changes to market behaviour, probably diminishing the effectiveness of traditional forecasting approaches, especially the ones assuming constant underlying process. Models based on parametric assumptions is suspected to underperform, either failing to capture longer-term dependencies or overfitting the noise present in high-frequency price data.

To address these limitations, this thesis investigates the use of Long Short-Term Memory (LSTM) neural networks, which are well-suited for modelling time series with complex temporal structures. The LSTM model is evaluated against a range of alternative approaches with varying levels of complexity and domain specificity, to benchmark its predictive performance and explore its interpretability in the context of electricity market forecasting.

## Motivation and Practical Applications

To understand the relevance of imbalance price forecasting, one must consider the structure and incentives of European electricity markets. Most energy retailers and producers participate in the Day-Ahead (DA) market, where electricity is traded based on forecasts of supply and demand. This mechanism enables grid operators and market participants to adjust production and consumption in advance, contributing to overall system stability.

However, real-world deviations from these forecasts are inevitable. Weather conditions may shift, transmission infrastructure can fail, and human or algorithmic errors occur. To address such discrepancies, the Intraday (ID) market allows participants to rebalance their positions closer to real time up to one hour before delivery. Yet, even with these adjustments, residual mismatches often persist.

The Balancing Market serves as the last line of defence, where the national Transmission System Operator (TSO) (Energinet in Denmark) activates reserves to maintain grid balance. These reserves are dispatched based on a merit order determined by marginal cost bids submitted in a separate auction. The resulting marginal cost dictates the imbalance price: the rate at which market participants are penalized or compensated for over- or under-delivery relative to their DA or ID commitments.

This imbalance price is notoriously volatile. It exhibits extreme price swings and fat-tailed distributions due in part to limited market liquidity, and by construction it is a response to unforeseen events. A significant number of potentially flexible assets remain excluded from participation due to stringent technical requirements and complex certification procedures. Consequently, while the need for fast-reacting reserves has grown, the pool of available participants has not expanded proportionally.
Recent regulatory changes to the imbalance price have further increased imbalance price standard deviation by an estimated 300% compared to last year, thereby raising the economic incentive for flexible assets to participate. In this context, the ability to forecast imbalance prices, particularly near or after the closure of the ID market, could allow flexible producers or consumers, in cooperation with their Balance Responsible Parties (BRPs), to adjust operations in response to live market conditions. This responsiveness could improve market efficiency, support grid reliability, and increase the value of previously underutilized flexibility.
This thesis investigates whether accurate imbalance price forecasting is feasible under current market conditions, and evaluates which modelling approaches are most effective in capturing the unique dynamics of this complex market segment.

# 2 Deep Learning: Long Short-Term Memory Networks

## 2.1 Recurrent Neural Networks (RNNs)

### Architecture

This section is based on Hochreiter and Schmidhuber [1997], and Shavlik [2015]. Recurrent Neural Networks (RNNs) is a large class of neural architectures specifically designed for modelling sequential data. Unlike traditional feedforward networks, RNNs maintain a hidden state that is propagated across time, enabling them to explicitly capture temporal dependencies. This recursive structure allows the network to retain information about previous inputs, thereby providing a form of internal memory. Formally, we can define a RNN as a function of the form:

$$f_\theta : (x_t, h_t) \mapsto (y_t, h_{t+1})$$

where $x_t$ is the input vector at time step $t$, $h_t$ is the hidden state vector at time step $t$, $y_t$ is the output vector at time step $t$, and $\theta$ represents the learnable parameters of the neural network. As a simple example we can look at the following classical RNN:

$$c^{(t)} = \sigma\left(w_r \cdot c^{(t-1)} + w_x \cdot x^{(t)}\right)$$
$$h^{(t)} = \sigma\left(w_c \cdot c^{(t)}\right)$$

Where, $x^{(t)} \in \mathbb{R}^n$ is input vector at time step $t$, $c^{(t)} \in \mathbb{R}^m$ is the cell (intermediate hidden) state, $h^{(t)} \in \mathbb{R}^m$ is the output, $w_x, w_r, w_c$ is learnable weight matrices that in union represent $\theta$, and the $\sigma(\cdot)$ non-linear activation function, e.g., sigmoid or tanh. This structure is visualized in Figure 2.1, where the RNN is "unfolded" across time. Each unrolled unit corresponds to the same RNN cell applied at different time steps, sharing parameters $w_x$, $w_r$, and $w_c$.

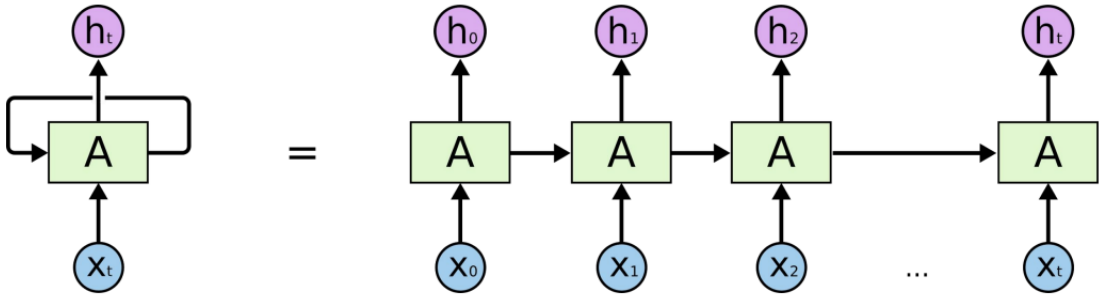


***Figure 2.1:*** *Unfolded architecture of a Recurrent Neural Network over three time steps. Each cell shares weights and passes its hidden state forward, capturing temporal structure in the input sequence.*

The recursive nature of RNNs enables them to process variable-length sequences and is particularly used for tasks like language modelling, speech recognition, and time series forecasting. However, the same temporal feedback that makes RNNs so applicable also has some inherent problems that arises when learning long-range dependencies, such topics will be discussed in the following subsections.

### Estimation and Training

The training of Recurrent Neural Networks (RNNs) is performed using a specialized version of backpropagation called *Backpropagation Through Time* (BPTT). This method accounts for the temporal dependencies in sequential data by unfolding the RNN across all time steps in the sequence. Each time step is treated as a layer in a deep feedforward network, with parameter sharing across all steps. For a more formal analysis let the output unit $k$'s target at time $t$ be denoted by $d_k(t)$, then by using mean squared error as the loss function, $k$'s error signal is given by:

$$\vartheta_k(t) = f_k'(\text{net}_k(t))(d_k(t) - y_k(t)),$$

where

$$y_i(t) = f_i(\text{net}_i(t))$$

is the activation of a noninput unit $i$ with differentiable activation function $f_i$,

$$\text{net}_i(t) = \sum_j w_{ij} y_j(t-1)$$

is unit $i$'s current net input, and $w_{ij}$ is the weight on the connection from unit $j$ to $i$.
Some nonoutput unit $j$'s backpropagated error signal is

$$\vartheta_j(t) = f_j'(\text{net}_j(t)) \sum_i w_{ij} \vartheta_i(t+1).$$

Several optimization strategies can be employed to compute the weight updates for $w_{jl}$, where $l$ is any unit connected to $j$, the most used are introduced here, based on Kingma and Ba [2017], Liu et al. [2020], and Hinton [2012]:

- **Gradient Descent.** In this project, i refer to the mini-batch variant, where gradients are accumulated over small batches of size $B$. The weight update is:

  $$w_{jl}(t) = w_{jl}(t-1) - \alpha \cdot \frac{1}{B} \sum_{n=1}^{B} \vartheta_j^{(n)}(t) y_l^{(n)}(t-1)$$

  Its foundational but its performance can be sensitive to batch size, therefore stochastic Gradient Descent is often preferred.

- **Stochastic Gradient Descent with Momentum.** Let $v_{jl}(t)$ be the velocity term associated with weight $w_{jl}$:

  $$v_{jl}(t) = \mu v_{jl}(t-1) - \alpha \vartheta_j(t) y_l(t-1)$$

  $$w_{jl}(t) = w_{jl}(t-1) + v_{jl}(t)$$

  where $\alpha$ is the learning rate and $\mu \in [0,1)$ is the momentum coefficient.

- **RMSProp.** Let $s_{jl}(t)$ be the running average of squared gradients:

  $$s_{jl}(t) = \rho s_{jl}(t-1) + (1-\rho)(\vartheta_j(t) y_l(t-1))^2$$

  $$w_{jl}(t) = w_{jl}(t-1) - \frac{\alpha}{\sqrt{s_{jl}(t)} + \epsilon} \vartheta_j(t) y_l(t-1)$$

  where $\rho \in [0,1)$ is the decay rate and $\epsilon > 0$.

- **Adam (Adaptive Moment Estimation).** Define first and second moment estimates $m_{jl}(t)$ and $v_{jl}(t)$ as:

$$m_{jl}(t) = \beta_1 m_{jl}(t-1) + (1-\beta_1)\vartheta_j(t)y_l(t-1)$$
$$v_{jl}(t) = \beta_2 v_{jl}(t-1) + (1-\beta_2)(\vartheta_j(t)y_l(t-1))^2$$

  Bias-corrected versions are computed as:

$$\hat{m}_{jl}(t) = \frac{m_{jl}(t)}{1-\beta_1^t}, \quad \hat{v}_{jl}(t) = \frac{v_{jl}(t)}{1-\beta_2^t}$$

  The weight update is then:

$$w_{jl}(t) = w_{jl}(t-1) - \frac{\alpha}{\sqrt{\hat{v}_{jl}(t)} + \epsilon}\hat{m}_{jl}(t)$$

  where $\beta_1, \beta_2 \in [0,1)$ are decay rates for the moment estimates.

Each optimization algorithm presents trade-offs in terms of speed, stability, and generalization. Traditional batch gradient descent provides a reliable gradient direction but is computationally prohibitive for large-scale or real-time scenarios. Stochastic Gradient Descent (SGD) with momentum addresses these issues by updating parameters more frequently and helping escape shallow minima, but it still requires careful tuning and can struggle with noisy or sparse gradients. RMSprop introduces adaptive learning rates, improving performance in non-stationary environments. Adam combines both momentum and RMSprop, achieving fast convergence.. However, empirical results Keskar and Socher [2017]often show that while Adam converges faster, SGD (with proper regularization and tuning) may yield better generalization in some deep learning tasks. The choice of optimizer thus depends on the specific characteristics of the dataset, model architecture, and training regime.

### Inherent Problems

While classical Recurrent Neural Networks offer a theoretically advantage for modelling temporal dependencies, they suffer from two major optimization challenges: the *vanishing gradient problem* and the *exploding gradient problem.* To see this we would look at how the error gets scaled though backpropagation. We start by looking at the local error flow as the global case immediately follows, for simplicity also assume that we use the traditional gradient decent. Suppose we have a fully connected net whose non input unit indices range from 1 to $n$. The error occurring at an arbitrary unit $u$ at time step $t$ is propagated back into time for $q$ time steps, to an arbitrary unit $v$. This will scale the error by the following factor:

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \begin{cases} f_v'(\text{net}_v(t-1))\, w_{uv} & q = 1 \\ f_v'(\text{net}_v(t-q)) \sum_{l=1}^{n} \frac{\partial \vartheta_l(t-q+1)}{\partial \vartheta_u(t)} w_{lv} & q > 1 \end{cases} \tag{3.1}$$

With $l_q = v$ and $l_0 = u$, we obtain:

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \sum_{l_1=1}^{n} \cdots \sum_{l_{q-1}=1}^{n} \prod_{m=1}^{q} f_{l_m}'(\text{net}_{l_m}(t-m))\, w_{l_m l_{m-1}} \tag{3.2}$$

(proof by induction). The sum of the $n^{q-1}$ terms $\prod_{m=1}^{q} f_{l_m}'(\text{net}_{l_m}(t-m))\, w_{l_m l_{m-1}}$ determines the total error backflow. If the product $\left| f_{l_m}'(\text{net}_{l_m}(t-m))\, w_{l_m l_{m-1}} \right| > 1$ across steps, then the backpropagated error grows exponentially with the number of time steps $q$, resulting in unstable learning, oscillating weights, or divergence. Likewise if this product is less than 1 at each step, the error shrinks

exponentially, resulting in the vanishing gradient problem. In this case, long-term dependencies cannot be learned effectively, as the signal fades before reaching relevant timesteps. Adjusting the learning rate does not improve performance since the ratio between long and short range errors remains the same, and would therefore only serve to make it more sensitive to recent observations. In the case with activation functions with derivatives in the range $(0,1)$ such as sigmoid or tanh this problem is amplified. The same problem persist if it is extend to the global case as can be seen by computing

$$\sum_{u \text{ output unit}} \frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)}.$$

To prevent vanishing error signals the recurrence must preserve magnitude. This leads to the requirement:

$$f_j'(\text{net}_j(t)) \cdot w_{jj} = 1.0.$$

Integrating the differential equation yields the condition

$$f_j(\text{net}_j(t)) = \frac{\text{net}_j(t)}{w_{jj}},$$

for arbitrary $\text{net}_j(t)$. This implies that $f_j$ must be linear, and the activation of unit $j$ must remain constant:

$$y_j(t+1) = f_j(\text{net}_j(t+1)) = f_j(w_{jj} y_j(t)) = y_j(t).$$

By letting $f_j(x) = x$ for all $x$, and setting the self-connection weight $w_{jj} = 1.0$. This setup is called Constant Error Carousel (CEC). In practical settings, unit $j$ is not isolated but receives inputs from, and sends outputs to, other units, this introduces two problems:

1. **Input Weight Conflict:** Consider an additional input connection to unit $j$ from unit $i$, associated with weight $w_{ji}$. Suppose that minimizing the total error requires activating unit $j$ in response to an input from unit $i$ and preserving this activation over an extended period. Given the linear nature of $f_j$ and the fixed input weight $w_{ji}$, the system faces competing objectives:

   - On one hand, the gradient may encourage an increase in $w_{ji}$ to facilitate the storage of important inputs by activating $j$.
   - On the other hand, the same weight must concurrently support the suppression of irrelevant or distracting inputs to prevent the premature deactivation of $j$.

   These opposing forces often result in conflicting gradient signals, and as consequence bad convergence.

2. **Output Weight Conflict:** Now assume unit $j$ maintains a stored representation of a past input. Consider a downstream connection to unit $k$, governed by weight $w_{kj}$. This single weight must simultaneously serve two conflicting purposes:

   - Permit the retrieval and transmission of stored information from $j$ to $k$ at appropriate times.
   - Prevent $j$ from interfering with $k$ when the stored information is not relevant to the current context.

   During early training, short-term error gradients often dominate and shape $w_{kj}$ accordingly. However, as training progresses and the network attempts to learn longer-term dependencies, the same connection may begin to introduce noise or disruption to units that had previously stabilized.

This problem is not exclusive to long time lags, even though they are significantly amplified in tasks requiring long-term memory retention. The longer the temporal gap, the more vulnerable stored information becomes to disruptions. In the next section a long short-term memory model is introduced that was developed to solve the gradient problem of the RNN.

## 2.2 Long Short-Term Memory Networks (LSTM)

This section is based on Hochreiter and Schmidhuber [1997], and Shavlik [2015]. Long Short-Term Memory (LSTM) networks are a specialized variant of Recurrent Neural Networks (RNNs) designed to address the challenges of learning long-term dependencies in sequential data. LSTMs reduces the gradient problems inherent in traditional RNNs by using a more advanced method to enforce constant error flow. The fundamental improvement in LSTMs is the introduction of a *cell state* $C_t$, which acts as a long-term memory buffer that flows through the network with minimal linear interaction. This memory is regulated by a set of gates, namely the *forget gate*, *input gate*, and *output gate* which control the flow of information into, out of, and through the cell. Figure 2.2 illustrates the internal architecture of an LSTM cell.

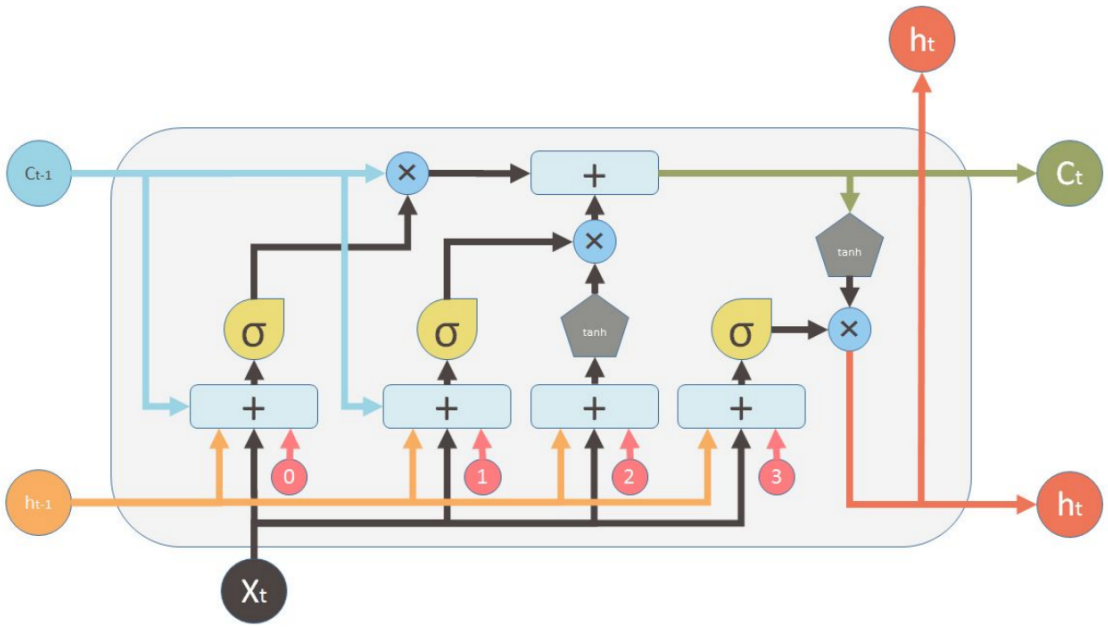


***Figure 2.2:*** *Internal structure of an LSTM cell showing the flow of information regulated by forget, input, and output gates, as presented in Shavlik [2015].*

At each time step $t$, the LSTM cell receives the current input $x_t$, the previous hidden state $h_{t-1}$, and the previous cell state $C_{t-1}$, and performs the following computations:

$$\phi^{(t)} = \sigma(w_{x\phi}x^{(t)} + w_{h\phi}h^{(t-1)}) \quad \text{(Forget gate)}$$
$$i^{(t)} = \sigma(w_{xi}x^{(t)} + w_{hi}h^{(t-1)}) \quad \text{(Input gate)}$$
$$o^{(t)} = \sigma(w_{xo}x^{(t)} + w_{ho}h^{(t-1)}) \quad \text{(Output gate)}$$
$$\tilde{c}^{(t)} = \tanh\left(w_{xc}x^{(t)} + w_{hc}h^{(t-1)}\right) \quad \text{(Candidate cell state)}$$
$$c^{(t)} = \phi^{(t)} \cdot c^{(t-1)} + i^{(t)} \cdot \tilde{c}^{(t)} \quad \text{(Updated cell state)}$$
$$h^{(t)} = o^{(t)} \cdot \tanh\left(c^{(t)}\right) \quad \text{(Updated hidden state)}$$

The forget gate $\phi^{(t)}$ determines which parts of the previous cell state $c^{(t-1)}$ should be discarded, while the input gate $i^{(t)}$ decides which new information is allowed into memory, thereby reducing the input weight conflict. Likewise the candidate state $\tilde{c}^{(t)}$ represents the potential new content to be integrated, and the output gate $o^{(t)}$ governs which parts of the cell state are exposed to the next layer or time step via the hidden state $h^{(t)}$, reducing the output weight conflict.
Training Long Short-Term Memory (LSTM) networks follows the same foundational approach as standard Recurrent Neural Networks, using *Backpropagation Through Time* (BPTT). However, due to the implementation of the gated cell state, propagation of the gradient flow is in principle more stable.
While a single-layer LSTM can capture temporal dependencies over long sequences, its representational power is still limited by the depth of the architecture. Deep LSTM networks solves this by stacking multiple LSTM layers vertically. The output hidden state $h^{(t)}$ from each layer at a given time step serves as input $x^{(t)}$ to the next layer, as shown in Figure 2.3.

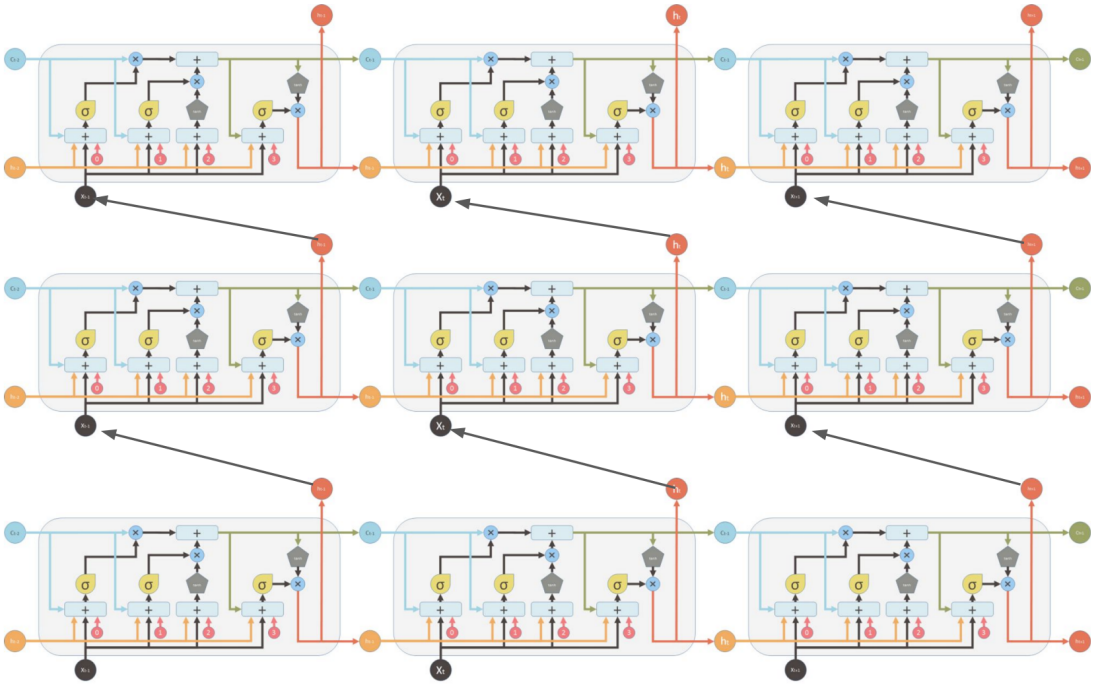

***Figure 2.3:*** *Stacked LSTM layers (Deep LSTM): each layer processes the sequence output from the layer below, enhancing model capacity for hierarchical temporal feature extraction, as presented in Shavlik [2015].*

In formal terms, for a deep LSTM with $L$ layers, the computations are defined recursively:

$$
\begin{aligned}
h_1^{(t)} &= \mathrm{LSTM}_1(x^{(t)}, h_1^{(t-1)}, c_1^{(t-1)}) \\
h_2^{(t)} &= \mathrm{LSTM}_2(h_1^{(t)}, h_2^{(t-1)}, c_2^{(t-1)}) \\
&\vdots \\
h_L^{(t)} &= \mathrm{LSTM}_L(h_{L-1}^{(t)}, h_L^{(t-1)}, c_L^{(t-1)})
\end{aligned}
$$

Each layer $\mathrm{LSTM}_\ell$ has its own set of parameters (weights and biases), and passes both the cell state $c_\ell^{(t)}$ and hidden state $h_\ell^{(t)}$ through time. The added expressiveness does come at the cost of large increase in parameters. As a result, LSTMs typically require large training datasets to achieve good

generalization, and their complex architecture makes them highly sensitive to hyperparameter settings.
Due to these limitations, the next section introduces several alternative models that can potentially compete with LSTMs in terms of interpretability, computational efficiency, or predictive performance.

## 2.3 Random Forest

### Architecture

This section is based on Breiman [2001].
Random Forest (RF) is a often used ensemble learning algorithm that operates by constructing a multitude of decision trees during training and aggregating their outputs. The method uses the principle of *bagging* (Bootstrap Aggregating), where each individual tree is trained on a different random subset of the training data sampled with replacement.
Each decision tree in the forest is built by recursively partitioning the feature space. However, to increase model generalization and reduce the correlation among trees, only a random subset of the input features is considered at each split.
For regression tasks, such as predicting electricity imbalance prices, the final output at a given time t $\hat{y}^{(t)}$ of the Random Forest is the average of the individual tree predictions:

$$\hat{y}^{(t)} = \frac{1}{N} \sum_{n=1}^{N} \hat{y}_n^{(t)}(x)$$

where $\hat{y}_n^{(t)}(x)$ denotes the prediction from the $n$-th tree, and $N$ is the total number of trees in the forest. A defining characteristic of RF training is that the trees are typically grown without pruning. That is, they are allowed to grow to their maximum depth (or until a stopping criterion such as a minimum number of samples per leaf is met). This overfitting at the individual tree level is counteracted by averaging over many diverse trees, which results in strong generalization performance at the ensemble level. Its architecture is also inherently parallelizable, which facilitates fast training on modern multi-core systems. Furthermore RF compared to other models, has a relatively low number of hyperparameters significantly reducing the complexity in application. The key hyperparameters governing the training of a RF include:

- The total number of trees in the forest,
- The maximum depth of each tree,
- The number of features considered for splitting at each node,
- The minimum number of samples required to split a node or form a leaf.

This parallel, low-bias, high-variance architecture, combined with bootstrapped resampling and feature randomness, makes RF particularly resilient to overfitting and well-suited for high-dimensional data with mixed signal-to-noise ratios. RF is well-suited for a wide range of prediction tasks, particularly when the input data is high-dimensional or noisy. By aggregating the outputs of many different decision trees, the model reduces variance and becomes less prone to overfitting than single-tree approaches. This ensemble structure makes RF robust to outliers and noise in the data.
One of the key practical benefits of RF is their ability to provide interpretable measures of feature importance. This combined with its fast training makes RF useful for both prediction but also for exploratory data analysis and feature selection.

Despite these advantages, RFs have inherent limitations, particularly when applied to time series data. Since each tree treats samples as i.i.d , RF do not inherently capture temporal dependencies or autoregressive structures. To apply them to sequential data, features such as time lags, rolling averages, or calendar effects must be engineered manually. Another limitation is that RF are not capable of extrapolation. That is, predictions for inputs outside the range of the training data are typically constrained to fall within the bounds observed during training. In regression tasks, this can result in outputs biased toward the global mean, particularly when inputs diverge from familiar patterns, as is often the case in the imbalance market.
In the scope of this project, Random Forests serve as a competitive baseline model. They are capable of capturing non-linear interactions between engineered features (e.g., weather, calendar variables, lagged prices), but lack the inherent temporal modelling capacity and ability to extrapolate that architectures such as LSTMs or ARIMA is able to. Nonetheless, their robustness, speed, and interpretability make them well suited for a comparison model.

## 2.4 XGBoost

This section is based on Chen and Guestrin [2016]. XGBoost (eXtreme Gradient Boosting) is a high-performance machine learning algorithm with architecture based on decision trees. Introduced by Chen and Guestrin [2016], XGBoost extends traditional boosting methods by incorporating advanced optimization techniques and regularization mechanisms, resulting in it being able to outperform many other ensemble models in both accuracy and training speed. The core idea of boosting is to build an additive model in a forward stage-wise fashion. At each boosting iteration $m$, a new tree $T_{[m]}$ is added to improve the prediction at time step $t$. The updated model at time $t$ after $m$ boosting rounds is given by:

$$\hat{y}^{(t)}_{[m]} = \hat{y}^{(t)}_{[m-1]} + T_{[m]}(x^{(t)})$$

Unlike traditional boosting methods, XGBoost uses a second-order Taylor expansion of the loss function, leveraging both gradient and curvature information to accelerate convergence. The objective function at iteration $m$ becomes:

$$\mathcal{L}_{[m]} \approx G^{(t)} T_{[m]}(x^{(t)}) + \frac{1}{2} H^{(t)} T^2_{[m]}(x^{(t)}) + \Omega(T_{[m]})$$

where $G^{(t)} = \frac{\partial \ell(y^{(t)}, \hat{y}^{(t)})}{\partial \hat{y}^{(t)}}$ is the gradient of the loss at time $t$, $H^{(t)} = \frac{\partial^2 \ell(y^{(t)}, \hat{y}^{(t)})}{\partial (\hat{y}^{(t)})^2}$ is the second-order derivative (Hessian), and $\Omega(T_{[m]})$ is a regularization term defined as:

$$\Omega(T_{[m]}) = \gamma J + \frac{1}{2} \lambda \sum_{j=1}^{J} w_j^2$$

Here $J$ is the number of leaves in tree $T_{[m]}$, $w_j$ is the weight assigned to leaf $j$, $\gamma$ penalizes model complexity via the number of leaves, and $\lambda$ controls the $\ell_2$-regularization on the leaf weights.
XGBoost also incorporates a lot of practical improvements such as shrinkage (learning rate), column subsampling (feature bagging), early stopping, and support for parallelized tree construction, all of which increases robustness, scalability, and applicability to a large variety of tasks. As a consequence of the large complexity and combination of methods implemented, extensive hyperparameter tuning is often needed, creating a use case for RF. Similar to RF XGBoost can also provides accurate feature importance metrics based on information gain, split frequency, or permutation importance. As is also the case with RF, XGBoost does not inherently model sequential or temporal dependencies, which introduces the need to manually engineer it as features. To ease said feature selection, and hyperparameter search, in all models, SHAP values are now introduced.

## 2.5 SHAP: SHapley Additive exPlanations

This section is based on Lundberg and Lee [2017], and Wikipedia contributors [2024]. SHAP (SHapley Additive exPlanations) is founded on Shapley values from cooperative game theory. In this context, each feature is considered a "player" contributing to the model's prediction. SHAP decomposes a model's output for a specific instance $x$ into additive feature contributions:

$$f(x) = \phi_0 + \sum_{i=1}^{M} \phi_i$$

Here, $\phi_0$ represents the expected model prediction over the training data (the model's bias), and each $\phi_i$ denotes the contribution of feature $i$ to the prediction for instance $x$. This is particular useful in linear models, especially when features exhibit multicollinearity, where traditional coefficient-based interpretations can be misleading. Shapley regression values address this by retraining the model on all possible subsets of features $S \subseteq F$, where $F$ is the set of all features. For each subset $S$, the model is trained both with and without feature $i$, and the difference in predictions is computed:

$$f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S)$$

The Shapley value $\phi_i$ for feature $i$ is then calculated as a weighted average of these differences across all subsets $S \subseteq F \setminus \{i\}$:

$$\phi_i = \sum_{S\subseteq F\setminus\{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} \left[ f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S) \right]$$

The Shapley value is uniquely defined by a set of properties that ensure a fair allocation of a total gain among cooperative agents. These properties make it highly suitable for attributing contributions in machine learning models. The first one being Efficiency ensuring that the total gain from the coalition is fully distributed among the players:

$$\sum_{i\in N} \phi_i(v) = v(N)$$

In the feature context this means that that all of the model output is accounted for in the feature attributions. The second one being symmetry that ensure that if two players $i$ and $j$ contribute equally to all coalitions, they receive the same Shapley value:

$$v(S\cup\{i\}) = v(S\cup\{j\}) \Rightarrow \phi_i(v) = \phi_j(v)$$

The third property is linearity that ensures any two games with value functions $v$ and $w$, the Shapley value of the combined game is the sum of the individual values:

$$\phi_i(v+w) = \phi_i(v) + \phi_i(w)$$

Also, for a scalar $a \in \mathbb{R}$:

$$\phi_i(a\cdot v) = a\cdot\phi_i(v)$$

This property ensures consistency under scaling and combination of games or models. The last property is the null player ensuring that only contributing features receive attribution:

$$v(S\cup\{i\}) = v(S) \quad \text{for all } S\subseteq N\setminus\{i\} \quad \Longrightarrow \quad \phi_i(v) = 0$$

The Shaply values has a wide range om applications, reflected in how Lloyd Shapley won a Nobel Memorial Prize in Economic Sciences, for it in 2012. In the context of this thesis it considered in the application of the following aspects:

- **Global feature importance**: Ranking features by their average absolute SHAP values can capture feature importance.
- **Local interpretability**: For individual predictions, SHAP allows for decomposition of the model output into positive or negative contributions from each feature, setting a basis for explainability.
- **Model evaluation**: By analyzing how SHAP values shift across different data regimes, changing market conditions can be identified.
- **Hyperparameter tuning**: SHAP can also be usedd to evaluate the sensitivity of model performance to various hyperparameters, possibly reducing the dimensionality of the tuning space.

In its general form, computing exact SHAP values requires evaluating all possible subsets. The naturally rapidly increasing the cost of computing making it unfeasible to check every subset for large dimensional problems. In practice this can be solved by sampling a representative subset of the whole dataset to then apply SHAP. This problem and the curiosity for other approaches, motivates the next section that represent other methods that is used in feature selection.

## 2.6 Feature Selection Techniques

In high-dimensional learning problems, feature selection is a essential part of improving generalization performance, reducing overfitting, and decreasing the computational load, often as a consequence of better signal to noise ratio. This section outlines several methods used in this thesis to select and evaluate features for imbalance price forecasting.

### 2.6.1 Recursive Feature Elimination (RFE)

This subsection is based on Hossain and Rahman [2023]. Recursive Feature Elimination (RFE) is a feature selection method that identify the most relevant subset of input variables by recursively training a model and pruning the least important features. The procedure operates as follows:

1. A base model (e.g., linear regression, decision tree, or random forest) is trained on the full feature set.
2. Feature importance scores—such as coefficients in a linear model or impurity reduction in a tree model—are computed.
3. The least important feature(s) are removed.
4. The model is retrained on the reduced feature set, and the process repeats until a desired number of features remain.

RFE is especially effective when used with tree-based models, which provide intrinsic measures of feature importance. It can be configured to stop at a fixed number of features or to evaluate model performance at each iteration using a validation metric such as cross-validated accuracy or mean squared error.
This approch has the advantage of explicitly optimizing model performance during the selection process. Which makes it well-suited for forecasting tasks where feature interactions and nonlinearity are important, as is often the case in energy market data.

### 2.6.2 LASSO

The Least Absolute Shrinkage and Selection Operator (LASSO), introduced by Tibshirani [1996], is a regularization technique that performs both coefficient shrinkage and variable selection within a single modeling framework. It modifies the ordinary least squares (OLS) loss function by adding an $\ell_1$-norm penalty on the model coefficients:

$$\mathcal{L}(\beta) = \sum_{i=1}^{n} \left( y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Here, $\beta_j$ denotes the coefficient of the $j$-th feature, $\lambda \geq 0$ is a regularization parameter controlling the strength of the penalty, and $n$ and $p$ are the number of observations and features, respectively. The $\ell_1$-penalty has the effect of shrinking many coefficients exactly to zero, particularly when $\lambda$ is large. This creates a sparse model that retains only a subset of the original features. Unlike ridge regression, which imposes an $\ell_2$-penalty and shrinks coefficients continuously, LASSO introduces a piecewise linear constraint that encourages sparsity.
LASSO is particularly useful in settings where the number of predictors is large relative to the number of observations, or when many features are expected to be irrelevant. As an embedded method, it integrates variable selection into the model training process, unlike wrapper methods that treat selection as a separate step.
In this thesis, LASSO is used primarily for linear benchmarking and feature pre-selection. While it is less effective at capturing non-linear interactions compared to tree-based methods, it provides a computationally efficient and interpretable baseline for identifying dominant linear relationships in the data.

### 2.6.3 Mutual Information

This subsection is based on Vergara and Estévez [2014]. Mutual Information (MI) is a measure from information theory that quantifies the amount of information one random variable contains about another. In the context of feature selection, it captures the statistical dependence between an input feature and the target variable, making it a valuable tool for identifying predictive relevance without assuming a specific model structure.
Formally, the mutual information $I(X;Y)$ between two random variables $X$ and $Y$ is defined as:

$$I(X;Y) = \iint p(x,y) \log\left( \frac{p(x,y)}{p(x)p(y)} \right) dx\, dy$$

where $p(x,y)$ is the joint probability density function of $X$ and $Y$, and $p(x)$ and $p(y)$ are their marginal densities. Intuitively, MI measures the reduction in uncertainty about $Y$ given knowledge of $X$, and it equals zero if and only if $X$ and $Y$ are statistically independent.
Unlike correlation-based methods, which only detect linear relationships, mutual information is capable of capturing both linear and non-linear dependencies.
In practice, MI is typically estimated using discretized bins or kernel-based approaches, especially when applied to continuous variables.

### 2.6.4 Tree-Based Importance

Tree-based models, such as Random Forest and XGBoost, inherently provide feature importance measures as part of their structure. These importance scores are derived from the internal decision-making process of the model and quantify the contribution of each feature to the model's overall predictive performance. Each time a feature is used to split a node, the improvement in the model's objective function is recorded. The cumulative contribution of each feature across all trees and splits yields its importance score.

Tree-based importance measures are efficient to compute and highly scalable, making them a practical tool for model interpretation and feature selection. However, they are also model-specific and may be biased toward features with many unique values or higher cardinality as shown in Zhou and Hooker [2020]. As such, they are best complemented with model-agnostic methods like permutation importance or SHAP for a more comprehensive understanding.

### 2.6.5 Permutation Importance

Permutation importance is a method for evaluating the relevance of input features by measuring the decrease in model performance when a feature's values are randomly shuffled. Unlike embedded methods, which rely on internal model parameters, permutation importance evaluates feature utility based on the actual impact of each feature on the model's predictions.
The key idea is to break the relationship between a feature and the target by permuting its values across the dataset while leaving all other features unchanged. The model's predictive performance is then re-evaluated on the perturbed dataset. If the model relies heavily on the permuted feature, performance will decline significantly; conversely, if the feature is unimportant, the impact will be minimal.
Formally, let $\mathcal{L}_{\text{orig}}$ be the loss on the validation set. For each feature $x_j$, a new validation set is constructed by randomly permuting the values of $x_j$, resulting in a perturbed loss $\mathcal{L}^{(j)}_{\text{perm}}$. The importance of feature $j$ is then quantified as:

$$I_j = \mathcal{L}^{(j)}_{\text{perm}} - \mathcal{L}_{\text{orig}}$$

This metric reflects the contribution of feature $j$ to the predictive accuracy of the model. A larger increase in loss corresponds to higher feature importance.
Permutation importance has several advantages: it is applicable to any predictive model regardless of structure, captures complex interactions, and directly reflects real-world impact on model performance. However, it can be computationally intensive, especially for large datasets or complex models, as it requires repeated inference passes.

## 2.7 Data Transformations and Normalization

Raw time series data from electricity markets often exhibit statistical properties, such as extreme skewness, and fat-tailed distributions, that can have a large influence on convergence. This is particular the case for algorithms that are sensitive to input magnitude, feature variance, or distributional assumptions. To address these issues, a variety of preprocessing techniques can be employed to transform and normalize the data prior to model training. We start by presenting some transformations, that are either common or / and is used directly in the application.

### 2.7.1 MinMax Scaler

The MinMax scaler transforms features by linearly rescaling them to a fixed range, typically $[0, 1]$. Given a feature $x$ with minimum value $x_{\min}$ and maximum value $x_{\max}$, the scaled value $x'$ is computed as:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

This transformation preserves the relative ordering and distances between values while aligning feature scales for models that are sensitive to magnitude, such as neural networks or distance-based algorithms. However, it is sensitive to outliers, which can compress the range of more typical values. Furthermore since it is should only be scaled based on the training set unseen data such as

negative values not present during training can result in the scaled output to fall below 0, causing inference issues.

### 2.7.2 Standard Scaler

The Standard scaler standardizes features by removing the mean and scaling them to unit variance. This transformation assumes the data are approximately normally distributed and is defined as:

$$x' = \frac{x - \mu}{\sigma}$$

where $\mu$ and $\sigma$ are the mean and standard deviation of the feature, respectively. This present a simple and interpretable scaling, but it is susceptible to outliers dominating the scaling.

### 2.7.3 Power Transformer (Yeo-Johnson)

To address skewed or heavy-tailed distributions—common in electricity price data, a non-linear power transformation can be applied to the target variable to enforce normality conditions. The Yeo-Johnson transformation is a generalization of the traditional Box-Cox transform that handles both positive and negative inputs. It stabilizes variance and promotes Gaussian-like behaviour, which is beneficial for models that assume or perform better with symmetrically distributed residuals.
This transformation is defined by a piecewise function that adapts to the sign of the input and a tunable parameter $\lambda$ learned during fitting. It is especially effective in reducing the influence of extreme values in imbalance prices. As a consequence this transform are applied to the classical ARIMA type models used in this thesis, ensuring the normality assumption.

### 2.7.4 Robust Scaler

The Robust scaler is designed for data with significant outliers. It centers the data using the median and scales it according to the interquartile range (IQR):

$$x' = \frac{x - \text{median}(x)}{\text{IQR}(x)}$$

This transformation is resilient to extreme values and is well-suited for volatile features, such as wind generation, imbalance volumes, or price spikes. It improves the robustness of models that are otherwise sensitive to distributional distortions. This transformation is used for RF, XGboost, and LSTM, due to the large number of outliers.

## 2.8 Regularization and Double Descent

Modern machine learning models, particularly those with high capacity, such as deep neural networks and ensemble methods, have the expressive power to fit highly complex functions. While this flexibility enables them to capture patterns in data, it also increases the risk of overfitting, especially in the presence of noise or limited data. To address this, regularization techniques are employed to constrain models ability to fit noise, thereby improving generalization. In addition to practical strategies, relatively recent observations such as the *double descent curve* provide a interesting view on implicit regularization especially for overparameterized models.

### 2.8.1 Bias-Variance Trade-Off and Classical U-Curve

Traditionally, the generalization error of a model is understood through the lens of the bias-variance decomposition. For a given input $x$, let $(x)$ denote the true underlying function and $\hat{y}(x)$ the learned prediction. The expected squared error can be decomposed as:

$$\mathbb{E}[(\hat{y}(x) - y(x))^2] = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise}$$

The bias measures the error introduced by approximating a complex function with to low capacity model, which leads to underfitting. The variance quantifies the sensitivity of the model to fluctuations in the training data, and as a consequence leading to overfitting. The irreducible noise arises from inherent randomness in the data generating process and cannot be eliminated by any model. This trade-off is often visualized as a U-shaped curve, where model complexity is plotted against generalization error. The classical view posits that increasing model complexity initially reduces bias and improves fit, but beyond a certain point, the variance dominates, and test error increases. Regularization techniques aim to control this trade-off by penalizing excessive flexibility in the model. In the next subsection some techniques is presented.

### 2.8.2 Regularization Techniques

The techniques used for regularization modify the learning objective by penalizing certain model behaviours, such as large weights, excessive depth, or neuron dependency encouraging more generalizable solutions.

**L1 and L2 Penalties.** Regularization is commonly applied through norm-based penalties on the parameter vector $\theta$. The two most widely used formulations are:

- **L1 Regularization (LASSO):** Adds a penalty proportional to the absolute value of the coefficients:

  $$\mathcal{L}(\theta) = \mathcal{L}_{\text{data}} + \lambda \sum_j |\theta_j|$$

  L1 regularization promotes sparsity by shrinking many coefficients to exactly zero, making it especially useful for feature selection in high-dimensional settings.

- **L2 Regularization (Ridge):** Adds a penalty proportional to the squared magnitude of the coefficients:

  $$\mathcal{L}(\theta) = \mathcal{L}_{\text{data}} + \lambda \sum_j \theta_j^2$$

  L2 regularization distributes weight shrinkage uniformly across all features, discouraging large parameter values and improving the stability of gradient-based optimization.

This form of regulation, reduce extreme weights and thereby reduces its ability learn noise in the training data. In the case for neuron dependency dropout is commonly used.

**Dropout.** In deep learning architectures, particularly in LSTM and feedforward neural networks, *dropout* is a stochastic regularization technique that randomly deactivates a fraction of hidden units during each training iteration. This prevents the network from relying too heavily on any particular activation pathway and forces the model to develop more robust internal representations. This technique has been shown to significantly improve generalization, particularly in deep recurrent architectures.

### 2.8.3 The Double Descent Phenomenon

This section is based on Schaeffer et al. [2023]. During the early stages of experimentation in this project, it was observed that the best-performing models consistently possessed a number of parameters far exceeding the number of training observations. Surprisingly, these overparameterized models performed better than the alternatives, even in the absence of explicit regularization techniques such as weight decay or dropout. This unexpected result motivated the inclusion of the double descent phenomenon in this thesis, as it offers interesting conclusions into the role of regularization and its interaction with model complexity.
Relatively recent results in statistical learning theory have shown that the classical U-shaped bias-variance trade-off does not fully explain the generalization behaviour of high-capacity models. Specifically, the double descent phenomenon contradicts the traditional assumption that test error must increase once a model becomes expressive enough to perfectly fit the training data, a point known as the interpolation threshold. As shown in Figure 2.4, test error initially decreases with increasing model complexity, consistent with the classical understanding that higher capacity reduces bias. However, at the interpolation threshold, test error often spikes due to increased variance and sensitivity to noise.

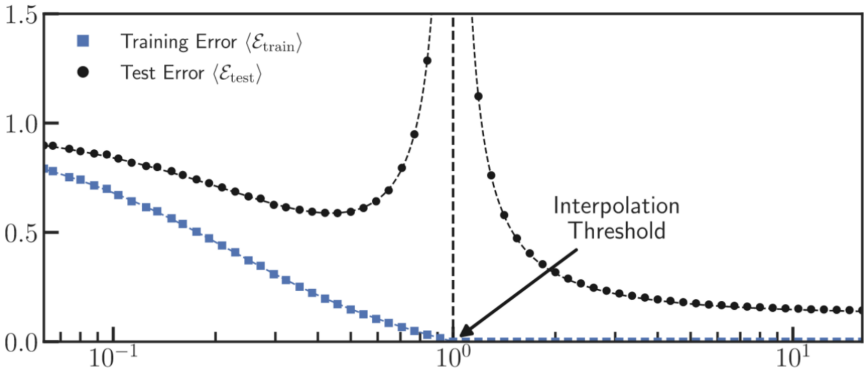


***Figure 2.4:*** *The double descent risk curve, illustrating traditional and modern regimes of model complexity, as shown in Schaeffer et al. [2023].*

Counter-intuitively, further increasing model capacity beyond this point can lead to a second decline in test error. This surprising improvement in generalization performance is commonly attributed to implicit regularization, which arises from the optimization dynamics of algorithms such as stochastic gradient descent and from architectural or initialization-related inductive biases. This observation influenced the modelling approach by explicit modelling of model complexity measured in parameters, and changes to hyperparameters such as maximum allowed epochs and patience in early stopping.

# 3 Hyperparameter Tuning

Hyperparameter tuning is particularly important when models are high-capacity, sensitive to initialization, or expensive to train. Unlike model parameters, which are learned from data during optimization, hyperparameters must be selected prior to training and govern aspects such as learning rate, regularization strength, model depth, and sequence length. Selecting appropriate hyperparameters can significantly impact a model's convergence behaviour and generalization performance. Poorly tuned hyperparameters may lead to underfitting, overfitting, or unstable training. Given the large and often non-convex nature of the hyperparameter search space, systematic search strategies are essential. This chapter presents two key tuning frameworks employed in this thesis: Hyperband, which is based on adaptive resource allocation via successive halving, and Bayesian Optimization, which uses a probabilistic surrogate model to guide exploration of the hyperparameter space.

## 3.1 Hyperband

Hyperband is a hyperparameter optimization strategy that allocates computational resources across a large number of candidate configurations. It extends the principle of successive halving, where weak configurations are rapidly discarded, allowing greater resources to be focused on promising candidates. This approach addresses the inefficiencies of grid and random search, particularly in high-dimensional or expensive-to-evaluate settings. Rather than treating all configurations equally, Hyperband dynamically adjusts the allocation of resources (e.g., number of training epochs or data samples) across configurations, prioritizing those that demonstrate early promise. This is done by using the concept of multiple brackets, each representing a trade-off between breadth (number of configurations) and depth (budget per configuration). Given a total resource budget $B$, a minimum allocation per configuration $r$, and a halving factor $\eta > 1$, the algorithm iteratively explores configurations using the following procedure:

$$s \in \left\{0, 1, \ldots, \left\lfloor \log_\eta \left(\frac{B}{r}\right) \right\rfloor\right\}$$

For each bracket $s$, Hyperband determines:

$$n = \left\lceil \frac{B}{r} \cdot \frac{\eta^s}{s+1} \right\rceil, \quad r_i = r \cdot \eta^{-i}, \quad i = 0, 1, \ldots, s$$

Where:

- $n$: number of configurations sampled in bracket $s$,
- $r_i$: resource budget allocated to surviving configurations at stage $i$,
- $\eta$: halving rate, typically chosen as $\eta = 3$.

At each iteration, only the top $1/\eta$ fraction of configurations are retained for the next round. This recursive filtering allows the algorithm to focus computational effort on the most promising candidates without requiring exhaustive evaluation of all possibilities. This results in Hyperband often being more sample-efficient than random or grid search, especially in high-dimensional hyperparameter spaces. Furthermore the successive halving structure makes it ideal to parallelize the computation decreasing the time significantly. As a consequence strong configurations that perform poorly in early stages may be prematurely discarded, particularly when early training dynamics are noisy or unstable.

## 3.2 Bayesian Optimization

This section is based on Frazier [2018]. Bayesian Optimization (BO) is a sample-efficient methodology for the global optimization of expensive and potentially noisy black-box functions. It is often used in scenarios where evaluating the objective function is computationally intensive, such as tuning hyperparameters of machine learning models, and where gradient information is unavailable, rendering derivative-based optimization techniques unsuitable. BO assumes that the objective function $f : \Lambda \to \mathbb{R}$, defined over a compact domain $\Lambda \subseteq \mathbb{R}^d$, is expensive to evaluate and lacks known analytical structure. Instead of evaluating $f$ exhaustively, BO maintains a surrogate probabilistic model of $f$, most commonly a Gaussian Process (GP), to capture both predictions and uncertainty. The GP prior over $f$ is specified as:

$$f(\lambda) \sim \mathcal{GP}(m(\lambda), k(\lambda, \lambda'))$$

where $m(\lambda)$ is the prior mean function, often set to zero, $k(\lambda, \lambda')$ is the covariance kernel, such as the squared exponential or Matérn kernel, which enables assumptions about smoothness and similarity. Then given a set of past evaluations $\mathcal{D}_t = \{(\lambda_i, f(\lambda_i))\}_{i=1}^t$, the GP posterior at a new point $\lambda$ yields:

$$f(\lambda) \mid \mathcal{D}_t \sim \mathcal{N}(\mu_t(\lambda), \sigma_t^2(\lambda))$$

where $\mu_t(\lambda)$ and $\sigma_t(\lambda)$ are the posterior mean and standard deviation. This posterior is then used to guide the selection of future query points. This is done by using an acquisition function $\alpha(\lambda \mid \mathcal{D}_t)$ that quantifies the expected utility of evaluating a candidate configuration $\lambda$ based on the current GP posterior. It allows BO to balance exploration (sampling where uncertainty is high) with exploitation (sampling near known optima). One of the most widely used acquisition functions is Expected Improvement (EI). It is defined by the expected value of the improvement over the current best observation:

$$\mathrm{EI}(\lambda) = \mathbb{E}\left[\max(0, f_{\text{best}} - f(\lambda))\right]$$

Assuming the GP posterior is $\mathcal{N}(\mu_t(\lambda), \sigma_t^2(\lambda))$ and $f_{\text{best}} = \min_{i \le t} f(\lambda_i)$, the EI can be computed in closed form as:

$$\mathrm{EI}(\lambda) = (\Delta(\lambda))\Phi\left(\frac{\Delta(\lambda)}{\sigma_t(\lambda)}\right) + \sigma_t(\lambda)\phi\left(\frac{\Delta(\lambda)}{\sigma_t(\lambda)}\right)$$

where $\Delta(\lambda) = f_{\text{best}} - \mu_t(\lambda)$, and with the standard normal PDF $\phi(\cdot)$ and $\Phi(\cdot)$ CDF. This formulation reveals the trade-off, where configurations with low predicted loss (high $\Delta$) or high uncertainty (high $\sigma$) yield high expected improvement. In summary with the posterior and the acquisition function defined the process Bayesian Optimization proceeds iteratively by the following algorithm:

**Algoritme 3.1** Pseudo-code for Bayesian optimization shown in Frazier [2018]

Place a Gaussian process prior on $f$
Observe $f$ at $n_0$ points according to an initial space-filling experimental design. Set $n = n_0$.
**while** $n \le N$ **do**
  Update the posterior probability distribution on $f$ using all available data
  Let $x_n$ be a maximizer of the acquisition function over $x$, computed using the current posterior distribution
  Observe $y_n = f(x_n)$
  Increment $n$
**Return:** either the point evaluated with the largest $f(x)$, or the point with the largest posterior mean

Several alternative acquisition functions have been proposed to address more complex Bayesian optimization scenarios. One notable example is the Knowledge Gradient (KG), which incorporates an estimates the expected improvement in the surrogate model's maximum after acquiring a new observation. KG is particularly well-suited for settings involving noisy evaluations. Another approach is Entropy Search (ES), which aims to reduce the uncertainty about the location of the global optimum by minimizing the entropy of its distribution. Additionally, multi-step acquisition functions have been introduced to account for the long-term effect of future evaluations, although they are typically computationally intensive. The choice of acquisition function should be based presence of noise, evaluation constraints, or opportunities for parallel function evaluations.

# 4 Application

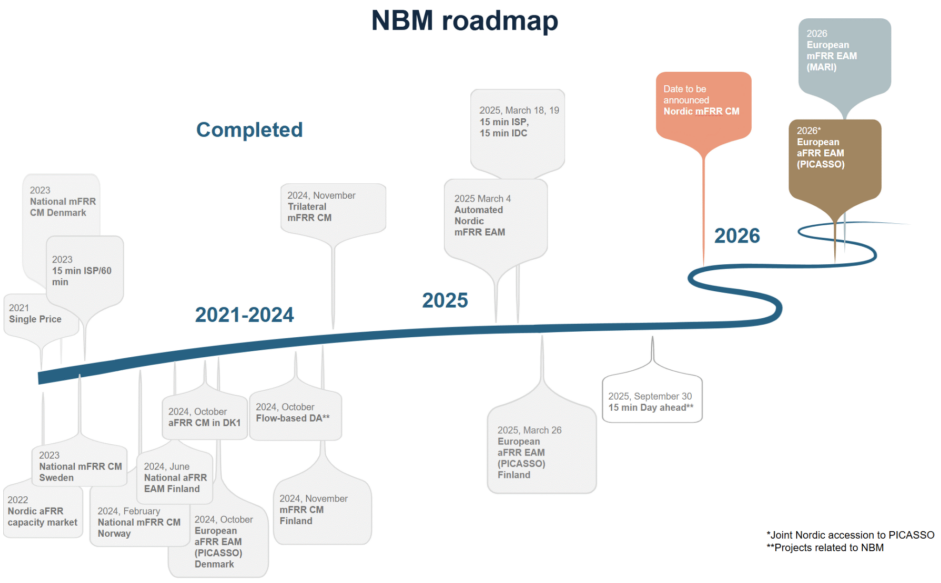


***Figure 4.1:*** *Nordic Balancing Model (NBM), roadmap of previous and upcoming market changes.*

This section outlines the structured pipeline used to collect, clean, engineer, and transform the dataset used for training and evaluation for a forecast of the imbalance price in the DK1 BZ. The problems encountered and what decisions was made to solve them is also discussed. We start by looking at the large number of policy changes that happened over this period, that changed the underlying process significantly, In 4.1 the systematic policy changes to the balancing market for the period from November 2021 to march 2025. Notably the 15 minutes ISP, mFRR CM, and mFRR EAM, is from a economic argument among the more influential policy changes, as these directly influence liquidity and volatility in the market. The changes is not exclusive to these, large non policy changes such as the war in Ukraine, renewable energy implementation and the intentional destruction of cables is not included among other things that also characterized this period. The decision about how far back to use data and how to split the test, train, and validation set, was based on several factors. Firstly the amount of important features that could be reasonably included, depends on the data size, and it was therefore suspected that the LSTM need several seasons (daily, weekly, and yearly) of data to correctly learn how to manage what information to store and discount in the state, this is the same reason for a 70-15-15 split. The imbalance price was before November 2021, not a single price but two prices depending on the direction of the imbalance, this sets a lower boundary date for the data to use, likewise the imbalance price design, was changed to also include aFRR in March 26, which for now sets and upper bound. Until January the 8th 2022 there where a high frequency of NA values in the relevant features, and this was therefore chosen as the start point. This is all summarized in 4.2 that show the imbalance price, the train test, validation split, and some of the significant structural changes.
As can be seen the period is highly non stationary a lot of shifts in the mean, variance, and just in general the distribution as can be seen on the frequency of the spikes, this is especially clear around January 2024. For this reasons some statistics on the distribution was calculated and can be seen in 4.1. The S&P 500 was included as stock returns is generally known for outliers, skew, and kurtosis, and would serve as a comparison, it should be noted that it is based on the close price but

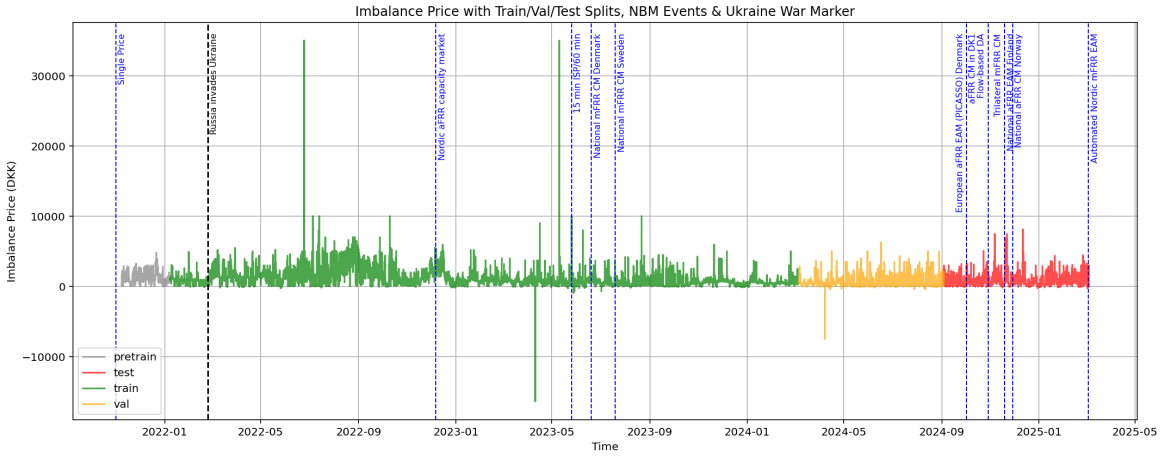


***Figure 4.2:*** *Imbalance price, with test, train, validation split, and marked known policy changes*

taken over the same period as the others. In this context the a data point is considered an outlier if it is less than the first quartile (Q1) minus 1.5 times the IQR, or greater than the third quartile (Q3) plus 1.5 times the IQR.

| Split | Mean | Std Dev | Skew | Kurtosis | IQR | Outliers | N | Outlier % |
|---|---|---|---|---|---|---|---|---|
| All | 925.6 | 1017.6 | 4.61 | 108.53 | 814.7 | 2360 | 29099 | 8.11 |
| Pretrain | 1169.3 | 735.2 | 1.22 | 1.87 | 837.6 | 70 | 1464 | 4.78 |
| Train | 1066.7 | 1137.3 | 4.56 | 103.58 | 928.6 | 1575 | 18905 | 8.33 |
| Val | 474.5 | 561.1 | 2.38 | 24.43 | 481.0 | 228 | 4364 | 5.22 |
| Test | 683.6 | 661.8 | 3.75 | 27.77 | 647.6 | 156 | 4366 | 3.57 |
| S&P 500 | 4678.1 | 697.3 | 0.59 | -0.85 | 1094.3 | 0 | 838 | 0.00 |

***Table 4.1:*** *Descriptive Statistics of Imbalance Price by Time Split and S&P 500*

The descriptive statistics in 4.1 offer some insight to the imbalance prices across different segments of the dataset. Overall, the full dataset exhibits a mean imbalance price of 925 DKK, with a standard deviation exceeding 1000 DKK and an extreme kurtosis of 108.5. The high skewness (4.61) further highlights the presence of a long right tail with large upward price jumps. In the pretrain period, covering the earliest phase of the data from the implementation of the Single Price model, the mean price rises to 1169 DKK, yet the standard deviation and higher-order moments are markedly lower. Skewness and kurtosis reduce to 1.22 and 1.87 respectively, suggesting a relatively stable period with fewer extreme events. This contrasts sharply with the training period, which exhibits the most severe market dynamics. With a standard deviation of 1137 DKK, skewness of 4.56, and kurtosis of 103.6, the training window reflects strong volatility and tail risk. Over 8% of observations in this period are considered outliers under the IQR rule.

The validation period reflects a transition to a more stable regime. The mean price declines sharply to 474 DKK, and both skewness and kurtosis fall significantly to 2.38 and 24.4, respectively. Finally, the test period continues this trend with a moderately higher mean of 684 DKK, but maintains relatively low variance and the lowest outlier ratio across all splits at just 3.57%. Despite this improvement, skewness (3.75) and kurtosis (27.77) remain well above Gaussian levels and normal stock returns. When benchmarked against S&P 500—typically the imbalance price demonstrates dramatically more extreme statistical properties. This comparison make the large requirements for the modelling approaches clear.

## 4.1 Data processing, feature selection, and scaling

Target and features was collected from Energi Data Service (EDS) using HTTP requests to public APIs. The first dataset called Regulating Balance Power data, contains the hourly imbalance quantities and prices. The second dataset is spot prices, thirdly a dataset with hard coded calender events such as holidays, special events, weekends, and a indicator that any of these was present to increase the frequency, so that events, like Christmas, that only happed 3 times could in some way be modelled. Lastly the dataset Forecast Wind and Solar Power which include the forecasted wind and solar production at different time points before the delivery. The time points include DA, ID, 5 hour, 1 hour, and currently marking the realized production. Some of this data included missing data, and some even included missing data for up to two weeks. The missing data in the target, could by inspection of the mFRR price be concluded to be and error in the distinction between 0 and NA on EDSs part. The missing data in the mFRR prices and volumes is a product of there not being any activation nor bids, this was fixed by setting the price equal to the spot, by argument from opportunity and marginal cost. The volume activated in that event was set to 0, as EDS normally do but was inconsistent. For the weather NA, a decision was made, that gaps of length less than 2 was interpolated by taking the last value. There where some gaps in some of the weather data columns that was around 2 weeks long, to fix this, the data from the other columns was used, in its place. After the data cleaning no further NA values where present and i could proceed with the data scaling. The data scaling was initially done my a min max scaler, this resulted in strange and bad results. Several factors influenced this, firstly since the train data was scaled between 0 and 1 the outliers in the training set dominated the scaling, resulting in bad convergence, secondly there was the risk that the minimum of the train data, was not the minimum over the whole data resulting in the model being unable to predict scaled negative prices, as these was never observed in the training. This was fixed by trying different scalers, firstly the standard scaler, but here again the outliers, dominated the scaling resulting in bad convergence, secondly was the robust scaler used that scale based on the IQR. This resulted in increased performance better convergence, and importantly better generalization.

### 4.1.1 Feature selection and engineering

Several engineered features was included in the selection, they can be distinguished in two categories, first where the ones that helps the model convergence, this was features with longer look back period than the LSTM, example of this is standard deviation over the last week, a 168 price lag to help it capture seasonality ect. The second kind of engineered features was done from a economic argument for example by the nature of the imbalance price, the difference between the actual weather and what was predicted should have higher impact than the actual weather. A regime id was also created based on the policy changes, there where 3 options considered on how to implement it firstly a traditional implementation using dummy variables, this increase the dimension significantly and was therefore discarded, secondly which was what was implemented was a single column with the consequence that the regimes would be difficult to capture correctly. Lastly embedding using a matrix to compare how alike the regimes is was considered, but was deemed out of the scope for this project. All features importance was tested, fairly fast it was observed that the difference in weather forecast showed better predictive power than raw weather data, and the later was therefore removed both with regards to dimensionality but also colinearity. The results for a feature important run are shown in 4.2.
The weights used to calculate the overall score was 0.1 for RFE, MI, and LASSO, it was 0.2 for tree-based, and permutation, and lastly 0.3 for SHAP. Firstly this was based on initial intuition, but later it was considered that a more appropriate weight distribution should be implemented. This is due to several reasons, firstly is that SHAP was implemented using RF so in principle SHAP should give better results than just using RF, which makes the later redundant. With respect to LASSO

| Feature | RFE | MI | Lasso | RF | PM | SHAP | Score |
|---|---|---|---|---|---|---|---|
| ImbalancePricelag168 | 8 | 13 | 14 | 8 | 11 | 6 | 9.1 |
| solarforecastdelta | 16 | 23 | 25 | 5 | 5 | 8 | 10.8 |
| mFRRDownActBallag1 | 11 | 21 | 22 | 6 | 8 | 9 | 10.9 |
| rollingstd168h | 10 | 4 | 20 | 14 | 12 | 10 | 11.6 |
| hour | 1 | 27 | 7 | 16 | 15 | 7 | 11.8 |
| ImbalancePricelag48 | 2 | 10 | 15 | 13 | 13 | 13 | 11.8 |
| rollingstdprevday | 13 | 12 | 17 | 10 | 10 | 12 | 11.8 |
| ImbalanceMWhlag1 | 5 | 16 | 8 | 12 | 17 | 11 | 12.0 |
| rollingstd48h | 17 | 8 | 23 | 9 | 7 | 14 | 12.2 |
| mFRRUpActBallag1 | 15 | 25 | 24 | 15 | 9 | 4 | 12.4 |
| ImbalancePricelag72 | 3 | 14 | 12 | 11 | 14 | 18 | 13.3 |
| timesincespike | 1 | 22 | 13 | 17 | 16 | 17 | 15.3 |
| imbalancestdprevday | 1 | 15 | 6 | 20 | 23 | 23 | 17.7 |
| ImbalanceMWhlag168 | 9 | 19 | 16 | 23 | 20 | 16 | 17.8 |
| meanimbalancelastweek | 1 | 7 | 11 | 22 | 25 | 22 | 17.9 |
| ImbalanceMWhlag48 | 12 | 18 | 19 | 18 | 18 | 20 | 18.1 |
| ImbalanceMWhlag72 | 7 | 17 | 16 | 21 | 19 | 21 | 18.3 |
| imbalancestd168h | 1 | 9 | 6 | 25 | 21 | 25 | 18.3 |
| dayofweek | 14 | 20 | 9 | 19 | 24 | 19 | 18.6 |
| imbalancestd48h | 4 | 11 | 18 | 24 | 22 | 24 | 19.7 |
| regimeid | 6 | 6 | 10 | 26 | 26 | 26 | 20.4 |
| mFRRDownActSpeclag1 | 18 | 26 | 26 | 27 | 27 | 27 | 25.9 |
| mFRRUpActSpeclag1 | 19 | 28 | 21 | 28 | 28 | 28 | 26.4 |

***Table 4.2:*** *Example of Feature Importance Rankings by Method and Combined Total Rank.*

and RFE they could potentially yield similar results, and is not guaranteed to contribute positively to the result. The results of the feature importance was used to select the 12 features used in the model can be seen in 4.3, along with some descriptive statistics, where it generally becomes clear that correct feature scaling is of large importance as the distribution and size for these variables are vastly different, especially for the weather features.

| Feature | Mean | Std Dev | Skew | Kurtosis | Min | Max | IQR | Outlier % |
|---|---|---|---|---|---|---|---|---|
| BalancingPriceDownlag1 | 828.61 | 812.17 | 1.32 | 19.95 | -16392 | 6982 | 706 | 7.33 |
| BalancingPriceUplag1 | 1036.62 | 1006.72 | 5.02 | 105.20 | -968 | 35000 | 767 | 8.96 |
| SpotPrice | 941.27 | 832.20 | 2.05 | 5.63 | -3277 | 6982 | 682 | 8.49 |
| mFRRUpActBallag1 | 15.71 | 51.93 | 5.40 | 43.82 | 0 | 1072 | 0 | 19.56 |
| Windforecastdelta | -85.84 | 303.36 | -0.72 | 1.57 | -1645 | 1314 | 335 | 4.26 |
| Hour | 11.50 | 6.92 | 0.00 | -1.20 | 0 | 23 | 12 | 0.00 |
| Solarforecastdelta | -6.23 | 82.89 | -1.27 | 21.11 | -1849 | 701 | 2 | 41.85 |
| ImbalancePricelag168 | 925.92 | 1017.96 | 4.60 | 108.38 | -16392 | 35000 | 817 | 8.10 |
| Rollingstd168h | 651.83 | 411.32 | 2.84 | 12.03 | 120 | 3312 | 313 | 8.58 |
| Rollingstdprevday | 484.80 | 471.68 | 7.09 | 93.19 | 14 | 8341 | 390 | 4.21 |
| mFRRDownActBallag1 | 22.99 | 63.90 | 5.20 | 42.61 | 0 | 1258 | 7 | 21.70 |
| ImbalanceMWhlag1 | -58.21 | 122.09 | -2.48 | 28.82 | -1857 | 1248 | 114 | 5.53 |

***Table 4.3:*** *Descriptive Statistics for Top Features in the Full Dataset*

## 4.2 Hyperparameter search

In the hyperparameter search both the Hyperband and the Bayesian approach is implemented. This is done by first implementing Hyperband to test a large range of parameters space, and then using RF in combination with sharp to determine the most influential parameters that is then fine tuned with the much smaller search space that favours the Bayesian approach. The decision for this approach was based on several factors primarily involving the computational setup, which is discussed in detail in appendix A and the exploration of the different approaches. The first part of the search results can be seen in 4.4, which was performed using Hyperband, it should be noted that due to nvidia GPU utilization some parameters such as batch size and recurrent dropout rate was fixed.

| Hyperparameter | Range | Best Value | SHAP (%) | Corr |
|---|---|---|---|---|
| lrrmsprop | [1.061e-05, 0.0009807] | 0.0008713 | 39.56 | -0.384 |
| lradam | [1.134e-05, 0.0009897] | 0.0006551 | 19.94 | -0.169 |
| Optimizer | [adam, rmsprop, sgd] | rmsprop | 7.85 | -0.109 |
| Unitslstm2 | [48, 192] | 128 | 5.86 | -0.112 |
| lrsgd | [1.016e-05, 0.0009694] | 4.91e-05 | 5.65 | 0.034 |
| Numlstmlayers | [1, 2] | 1 | 4.81 | 0.002 |
| Unitslstm1 | [48, 192] | 64 | 3.13 | 0.118 |
| Dropoutrate | [0.3, 0.5] | 0.3 | 2.17 | 0.042 |

***Table 4.4:*** *Hyperparameter Summary with Normalized absolute SHAP Importance (%)*

It was tested if any of the optimizers was structurally worse, which in the case of categorical variables could have made it non productive to optimize further and would only increase the search space. The SHAP value shows that the primary hyperparameters in this setup was the learning rates and the optimizer, which was used is then selected as the hyperparameters in the Bayesian setup, the results can be seen in 4.5.

| Results | Hyperband | Bayesian Opt. |
|---|---|---|
| optimizer | rmsprop | adam |
| lr | 0.0008713 | 0.0004770 |
| Number of Trials | 754 | 60 |
| Max Epochs | 180 | 100 |
| Validation MAE | 0.29470 | 0.27299 |

***Table 4.5:*** *Comparison of Hyperband vs Bayesian Optimization Tuning Results*

It should be noted here that when it was observed that occasionally it hit the maximum limit of the number of epochs in the Hyperband approach. Based on this the maximum epochs for the Hyperband was increased, as training time was acceptable. The maximum limit on epochs for the Bayesian approach, was kept. It could be argued that there should be same max epoch limit for both to compare across, but due worse GPU utilization, and slower training speed for each trial in the BO approach this was decided against. Again see A for a detailed discussion on the computational setup, training time, and convergence. Despite less trials and less max epochs the Bayesian approach to fine tune a subset of hyperparameters seemed to reduce the validation MAE. It should also be noted that using such a large number of trials could potential introduce data leakage as such a number of trials increase the probability that the configuration just randomly fits the validation set, if this is the case for the Hyperband results the Bayesian approach "improving"

on a almost already selected configuration, would only make this worse. To reduce the probability of this happening, i tried the obtained configuration with a different train and validation split, but with similar performance with different splits, no further investigations where made.
During the initial implementation of the LSTM model, the model was not configured to be stateful, meaning it did not retain its internal state across batches. While this made certain aspects of the training process simpler, such as allowing randomized batch ordering and handling breaks in the data caused by missing values, it also introduced a significant limitation. Specifically, resetting the LSTMs state at the end of each batch prevented the model from learning dependencies that spanned across batches.
Attempting to mitigate this by increasing the batch size led to further complications. Larger batches altered the convergence behaviour and significantly increased memory consumption. As a result, the stateless LSTM performed unreasonable poorly. To address this, a stateful LSTM was implemented, in which the internal state is preserved across batches within an epoch but reset between epochs. This change enabled the model to capture longer-term dependencies more effectively, resulting in better performance.

## 4.3 Forecast procedure and Comparison Models

To benchmark the LSTM model, it was tested against classical and tree-based methods. All models was trained under the same test train and validation split. Forecasts was compared using a one hour ahead rolling window with periodic retraining on the whole dataset every 168 hours.
The classical models used a generalized version of the Box-Cox, namely the Yeo-Johnson transformation to handle negative data and to make the data more normal for the OLS assumptions. Rest of the models used robust scaler to reduce the influence of outliers. Two families of time series models are applied firstly some naive Univariate autoregressive models (AR(1), AR(24), and AR(168)) the second family of models was ARIMA (ARIMA(3,0,5), and ARIMA(2,0,1)) selected by AIC and BIC using the python package AUTOARIMA.
The tree-based methods used in this study were Random Forest (RF) and XGBoost, both implemented in Python. Due to time constraints, no extensive hyperparameter optimization was performed; however, a set of reasonable defaults was selected for both models, after some attempts. The Random Forest model was configured with 300 estimators and a maximum depth of 8. The XGBoost model also used 300 estimators, with a maximum depth of 6, a learning rate of 0.05, and both row-wise and column-wise subsampling set to 0.8. These settings were found to provide stable performance without unreasonable overfitting (monitored by training vs validation loss), while maintaining manageable computational cost.

### 4.3.1 Co-integration

After observing no consistent ability to capture directional accuracy in any models except the ensemble methods (Random Forest and XGBoost), classical models were tested with and without differencing. However, differencing did not improve their directional explanatory power. It was suspected that this limitation could be due to strong non-stationarity, due to both deterministic, and stochastic seasonality as can clearly be seen in B.1. In an attempt to test for simple unit roots there was at first performed two traditional tests namely the KPSS and ADF, but with both rejecting the Null leaving it inconclusive. The same "non" conclusion also characterize the literature as explained in Wang and Tomek [2004]. However to check the possibility for seasonal unit root an attempt was based on the recommendation in Ronderos [2019]. The method is to use a periodogram, to first identify the type of seasonality. The result can be seen as the middle plot in 4.3. Based on the comments in Ronderos [2019] both the Spot price and the imbalance price exhibits what looks like stochastic seasonality, to test if this is the source of a non-stationarity

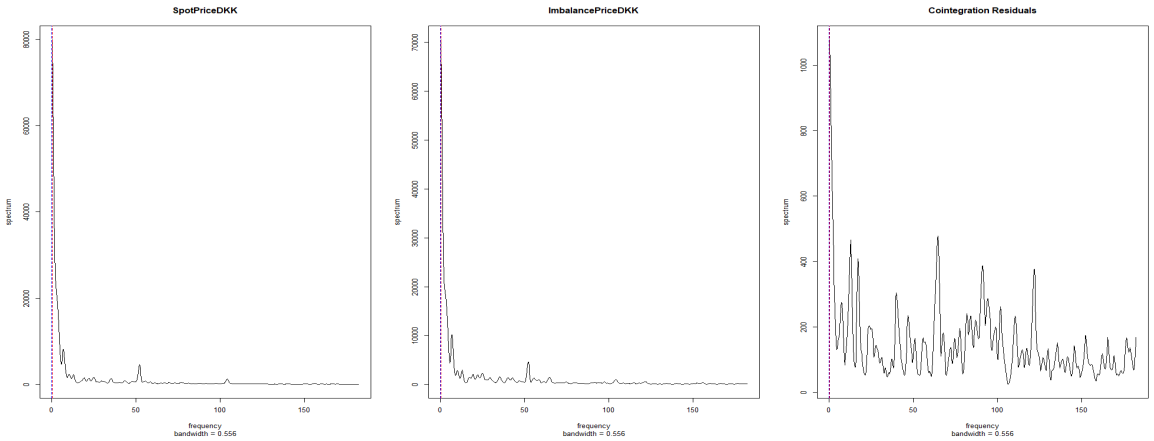


***Figure 4.3:*** *Periodogram over 200 hours for the DA Spot Price, the Imbalance Price, and the co-integrated residuals.*

unit root, the HEGY test was recommended but deemed out of the scope for this project. The conclusion for the unit root will therefore be based on economic argument around the persistent effects that characterize unit roots. The simplest of such arguments can be done from a policy perspective, where the Danish TSO (Energinet) implements regular adjustments (approximately at a frequency of once a week) to the policy of the balancing market, such as volumes bought in the capacity market, deadband policy (how much imbalance is tolerated) connected bidding areas changes to policy ect. that should all in principle result in non mean reverting shocks. Based on this it is seemingly reasonable to assume that a form of unit root is present, and will therefore for the rest of this thesis be assumed so. To fix this potential issue co-integration between relevant series was considered, and the spot price was deemed a potential candidate. From an Efficient Market Hypothesis (EMH) standpoint, changes in the imbalance price relative to the spot price should primarily be based on new information arriving in the intermediate period and the associated risk premium. In other words, the difference between the imbalance price and the spot price should, by economic reasoning, co-integrate (assuming unit root is present). This hypothesis was tested using the Engle-Granger co-integration test, and the results are shown in Table 4.6.

| **Test / Metric** | **Value** | **Remarks** |
|---|---|---|
| *Engle-Granger Co-integration Test* | | |
| Test Statistic | -9.9347 | Highly significant |
| P-value | $3.63 \times 10^{-16}$ | Reject null hypothesis |
| Critical Value (1%) | -3.96 | |
| Critical Value (5%) | -3.41 | |
| Critical Value (10%) | -3.12 | |
| *OLS Regression: ImbalancePriceDKK on SpotPriceDKK* | | |
| Intercept ($\alpha$) | 18.3969 | Statistically significant |
| Slope ($\beta$) | 0.9660 | Strong positive relationship |
| R-squared | 0.616 | Moderate explanatory power |
| Observations | 27,635 | Full dataset |
| **Test of** $\beta = 1$ | T = -7.3972, P < 0.0001 | Reject null (significant deviation) |

***Table 4.6:*** *Cointegration Analysis Between SpotPriceDKK and ImbalancePriceDKK*

The Engle-Granger test strongly rejects the null hypothesis of no co-integration, with a test statistic of -9.93 and a highly significant p-value. This confirms that the day-ahead spot price and the imbalance price are cointegrated, and that the spread between them is mean-reverting. From an economic perspective, this is consistent with EMH, as the spot price reflects the market's risk-adjusted expectations based on available information, while the imbalance price adjusts for deviations near real-time. Likewise this is also visually confirmed in B.1 and by the 4.3. Especially the periodogram shows significant change in process dynamics. Interestingly the estimated co-integration relationship shows that the imbalance price responds to changes in the spot price with a slope coefficient ($\beta$) of 0.9660, implying a nearly one-to-one long-run adjustment. However, a formal hypothesis test reveals that this coefficient is statistically different from unity (T = -7.3972, $p < 0.0001$), suggesting a persistent spread, likely representing a structural risk premium. In summary, the co-integration analysis validates the use of the spot price as a stable long-run anchor for modelling the imbalance price, which is utilized across all implemented models.

## 4.4 Forecasting Results

| **Model** | **Setting** | MAE | RMSE | SMAPE | $R^2$ | MBE | DA | Out-MAE | Non-MAE | Out-MBE | Non-MBE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LSTM | Without | 229.44 | 498.23 | 41.10 | 0.437 | -80.64 | 0.521 | 1986.61 | 178.22 | -1955.35 | -26.00 |
| | With | 203.07 | 489.20 | 37.06 | 0.456 | -39.68 | 0.637 | 1720.93 | 158.83 | -1639.97 | 6.96 |
| RF | Without | 221.18 | 478.52 | 40.39 | 0.481 | -11.37 | 0.585 | 1687.38 | 177.86 | -1601.58 | 35.63 |
| | With | 207.71 | 480.81 | 37.68 | 0.476 | -8.10 | 0.615 | 1555.57 | 167.88 | -1418.35 | 33.57 |
| XGBoost | Without | 222.86 | 472.92 | 40.27 | 0.493 | -5.14 | 0.568 | 1665.06 | 180.24 | -1581.21 | 41.43 |
| | With | 217.91 | 477.49 | 39.21 | 0.483 | 0.37 | 0.592 | 1647.89 | 175.65 | -1510.69 | 45.02 |
| ARIMA(3,0,5) | Without | 286.23 | 501.87 | 50.21 | 0.429 | 37.22 | 0.499 | 1711.77 | 244.11 | -1590.44 | 85.32 |
| | With | 217.01 | 459.87 | 41.87 | 0.520 | -43.86 | 0.607 | 1530.37 | 178.20 | -1379.11 | -4.40 |
| ARIMA(2,0,1) | Without | 286.15 | 519.32 | 51.08 | 0.388 | -25.83 | 0.497 | 1779.06 | 242.03 | -1656.94 | 22.37 |
| | With | 212.41 | 461.22 | 39.98 | 0.517 | -31.24 | 0.620 | 1516.47 | 173.88 | -1360.96 | 8.06 |
| AR(168) | Without | 266.15 | 487.39 | 47.92 | 0.461 | -2.72 | 0.509 | 1723.03 | 223.10 | -1620.76 | 45.10 |
| | With | 219.95 | 461.51 | 42.20 | 0.517 | -32.15 | 0.581 | 1530.51 | 181.22 | -1368.09 | 7.33 |
| AR(24) | Without | 269.07 | 493.02 | 47.67 | 0.449 | 5.11 | 0.512 | 1718.05 | 226.25 | -1594.49 | 52.38 |
| | With | 214.88 | 460.84 | 40.85 | 0.518 | -36.61 | 0.611 | 1535.21 | 175.87 | -1373.61 | 2.90 |
| AR(1) | Without | 278.48 | 511.03 | 49.58 | 0.408 | 43.40 | 0.507 | 1672.46 | 237.29 | -1428.50 | 86.89 |
| | With | 215.28 | 462.23 | 41.14 | 0.515 | -41.66 | 0.628 | 1536.48 | 176.24 | -1377.59 | -2.18 |

***Table 4.7:*** *Model Performance With and Without Co-integration (All Metrics)*

The table in 4.7 compares the results of the different set of models implemented, with and without a co-integration based transformation of the target variable where spot prices are subtracted. One of the most notable outcomes from the performance table is the consistent improvement observed in all models after co-integration is applied, particularly in terms of MAE, SMAPE, and Directional Accuracy. The only exception being the tree models RMSE and $R^2$, that surprisingly got worse. The likely cause of the relatively less improvement for the tree models can be found in the nature of the models as the other models uses the time series directly, the tree models only do it implicit, by lagged features. It should be noted that different levels of differencing was tested on the AR based models without improved performance, and the AUTOARIMA suggested no differencing with the order (2,0,1), after co-integration the AUTOARIMA suggested (3,0,5).
In the non-cointegrated setting, the ARIMA models surprisingly performed worse than the simpler AR models across most metrics. This result may reflect in a sense "unnecessary" added complexity that is still unable to capture the true underlying process, and therefore over fitted short-term fluctuations, caused by other reasons. In contrast, the AR models, particularly AR(1) and AR(24), were better suited to capturing short-term autocorrelation patterns without introducing unnecessary parameters, due to their naive approach.
After introducing co-integration, the performance landscape shifted. ARIMA models improved significantly, as did the AR models, but the relative gains were not uniform. Notably, AR(168), which incorporates a week of hourly lags, underperformed compared to AR(1) and AR(24). This suggests

that once the long-run relationship between spot and imbalance prices is accounted for through co-integration, deeper lags may introduce more redundancy than value. The poor performance of AR(168) can likely be attributed to model overcomplexity, high multicollinearity among lags, and a diluted signal-to-noise ratio, again fitting to noise caused by other things.
In this context, the AR(1) model still performed surprisingly well after co-integration was introduced. This outcome may be tied to the short-memory behaviour of the system, or in other words, "stickiness" where market reactions and balancing decisions unfold over very short time periods. Co-integration likely acts as a filter, removing much of the noise embedded in the spot price and isolating the structural deviations. This filtered signal appears particularly advantageous for simple models like AR(1), which are highly sensitive to dominant directional trends and benefit more when irrelevant variance is reduced. Compared to the tree models, co-integration seems to benefit models that are more sensitive to raw noise, non stationarity or overfitting, like LSTM and the classical models.
Regarding LSTM, although the spot price was already included as a feature in the non-cointegrated version, the relationship between the spot price and imbalance price is complicated, involving a direct anchoring effect on the price, but also how it changes the marginal price of additional production close to delivery. The co-integration transformation helped the model capture the anchoring role of the spot price, and by keeping it as a feature isolating the effect on the marginal price change. In 4.4 the predicted vs actual for the LSTM with co-integration can be seen, showing reasonable ability to predict spikes and changing pattens.

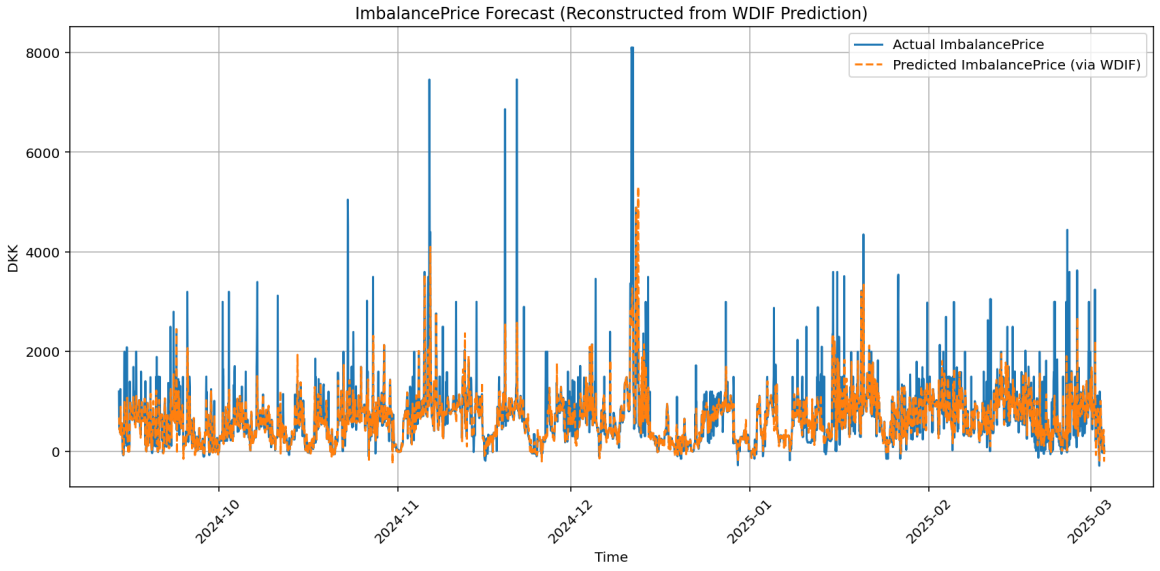


***Figure 4.4:*** *Predicted against actual, in the co-integrated LSTM*

XGBoost already had low MBE in the non-co-integrated setting, indicating low bias, but it struggled with distinguishing between outlier and non-outlier MBE. This may relate to learning past structure "too well," which is less helpful in a market prone to structural and policy-driven shifts. Which does not seems to be as detrimental to models like RF or AR. This could be the general case and the reason surprisingly strong performance of simpler models. As discussed previously indicators for these changes could be a solution but with sufficiently many shifts the signal to noise ratio was deemed unfavourable. Embedding would be a strong solution especially considering the ability to model similarity across regimes which will be highly applicable in known future changes, nevertheless it was deemed out of the scope for this project.

# 5 Conclusion

The Danish balancing market is becoming more volatile and less predictable due to structural changes, increased renewable penetration, and regulatory reform. This thesis set out to test whether forecasting imbalance prices under these new market conditions was even feasible and to what extent modern machine learning methods like LSTM could offer an advantage. The results of this process can be seen in the table 4.7. The highlight several patterns worth mentioning, especially in terms of how different models respond to the co-integration transformation.

One of the most consistent findings is that all models improve when trained on co-integration (assuming unit root) between the spot price and the imbalance price. This is most clearly seen in MAE, SMAPE, and DA, and to a lesser extent in MBE. However, the improvement is not equally strong across all models. The tree-based models (RF and XGBoost) surprisingly show a slightly worse RMSE and $R^2$. A likely explanation is that these models already model non-linear interactions well, and introducing a pre-transformed target (spot-subtracted imbalance price) doesn't necessarily help and potentially even adds noise or removes signal they already picked up on through lagged spot features.

In the non-cointegrated setting, ARIMA models actually perform worse than their AR counterparts. This is surprising, but probably points to unnecessary complexity. ARIMA tries to model short-term dynamics in the residual noise, but without differencing or strong trend structure, it doesn't help and might even overfit. The AR models, especially AR(1) and AR(24), do relatively well by just capturing short-term autocorrelation. Their simplicity actually works in their favour here.

Once the co-integration transformation is applied, things shift. ARIMA models improve a lot—as expected when the underlying series becomes more stable and predictable. But AR(168), which should in theory benefit from a full weekly seasonality, now performs worse than AR(1) and AR(24). This likely comes down to the fact that the co-integration already handles long-run structure, so adding a long lag only adds noise and collinearity.

AR(1) is a bit of a standout. Even after co-integration, it performs surprisingly well. This suggests that the system has short memory, or what could be called "stickiness." Market reactions happen quickly, and once you remove the long-term drift (via co-integration), what's left is mostly short-range variation that suits AR(1).

For the LSTM, the benefit of co-integration is more complex. Although spot price is already used as a feature in the non-transformed setup, it's difficult for the model to learn both the anchoring relationship and the marginal adjustment signal at the same time. Subtracting the spot price makes the target focus more on the short-term deviations, while still keeping the spot price as a feature to help model its influence on marginal activation price. This separation likely helps the signal to noise ration and therefore improve, both accuracy and spike prediction, as seen in Figure 4.4.

The tree models are a bit more resistant to this transformation. XGBoost especially already had low MBE, meaning low bias, but struggled with separating outlier and non-outlier bias. This is likely due to insufficient hyperparameter optimization. Another possibly contributing reason is that it learns structure "too well" which works in stable regimes, but not in a market that changes structurally. RF seems less affected, likely because it's less aggressive in fitting residual noise, and the less sensitivity to hyperparameter tuning.

All of this points to models like LSTM, AR, and ARIMA that are sensitive to non-stationarity, raw noise, or risk of overfitting, benefit the most from co-integration. Tree models already encode non-linear structure and lag interactions, so the gain is still clearly present but more limited and even slightly harmful in RMSE.

It's also worth noting that the ARIMA orders change after transformation. Without co-integration,

the best ARIMA was (2,0,1); with co-integration, it was (3,0,5). This again shows that stabilizing the series changes the nature of the dynamics and what the model should focus on.
In general, this confirms that using co-integration as a pre-transformation is a valid and effective way to improve forecastability. The spot price acts as a long-run anchor, and subtracting it simplifies the dynamics the models need to learn. Whether for neural nets or classical models, it improves signal-to-noise and helps reduce the risk of overfitting to short-term regime-specific noise.
From a broader perspective, the results for LSTM showed that accurate forecasting is possible, especially when structural relationships like the one between spot and imbalance prices are explicitly modelled. LSTM showed strong improvements under co-integration and outperformed classical models in multiple dimensions, especially for directional accuracy and capturing extreme price spikes. At the same time, the surprising strength of simpler models like AR(1) after transformation suggests that under the right pre-processing, even low-complexity models can perform competitively.
This highlights that effective forecasting is less about model complexity in isolation, and more about aligning model structure with the economic and statistical properties of the underlying data. The co-integration transformation directly addresses this need and improves the signal quality for all models. Given the challenges outlined in the introduction, non-stationarity, illiquidity, and regime shifts, the approach taken here offers a reasonable path for improving forecast performance with the goal of using flexible assets to act post-Intraday market closure.

# Appendix

# A Computational configuration and optimization

## A.1 Computational Setup and cuDNN Optimization

The project was developed using a three computational setup. First, a local laptop was used for fast code testing. Second, a virtual machine (VM) running Windows Server was accessed remotely. This machine was equipped with an AMD EPYC 9124 16-core 3.0 GHz processor and 64 GB of memory, but no GPU. It was primarily used for high-speed CPU tasks such as feature engineering, classical model evaluation, and early Hyperband optimization runs.
The third machine was a physical Windows workstation equipped with an 11th Gen Intel(R) Core(TM) i7-11700 CPU at 2.5 GHz, 16 GB RAM, and a dedicated NVIDIA RTX A2000 (12 GB VRAM) GPU. To enable GPU-accelerated training in TensorFlow, the Linux-based Python environment was set up inside WSL (Windows Subsystem for Linux). The memory and swap allocation were adjusted several times to reduce crashes and improve performance during training.

## A.2 Hyperparameter Constraints in cuDNN Stateful LSTMs

To take full advantage of the GPU, the LSTM implementation was configured to force cuDNN compatibility. This meant using a restricted configuration with activation function fixed as tanh, and recurrent activation fixed as sigmoid, and no recurrent dropout. Although it reduced flexibility, this decision significantly boosted training speed and allowed for better hyperparameter exploration, along with better scalability, generally leading to better results.
The combined Hyperband and Bayesian setup was motivated by the possibility in future setups to split optimization tasks between the CPU-optimized VM and the GPU-accelerated workstation. Hyperband could be run on the VM to search broadly, while Bayesian Optimization could be applied on the GPU machine to fine-tune parameters. This division could allow for a more efficient use of both compute environments, especially the strong CPU no GPU setup, which would otherwise be underutilized. The implementation of this could be done in python using ray, allowing rescourse allocation and parallelzation across different machines.
To improve temporal learning performance, a stateful LSTM was implemented. This allowed the model to carry hidden states between batches—essential for learning long-term dependencies. However, this forced the fixed batch sizes, which was chosen to be 32 based on preliminary results. As with the other limitations to it directly reduced the search space reducing the amount of trails needed but at the cost of flexibility. While this setup introduced some rigidity in model design and training, it was necessary to stabilize training and make full use of the GPU via cuDNN. The choice to go with this approach was made after observing significantly worse performance in stateless variants.

## A.3 Memory Optimization and Management

Memory usage became a bottleneck during training, especially with long lookback windows and larger models. To reduce this impact several methods was implemented namely mixed precision, and downcasting.

TensorFlow's mixed precision policy was used allowing core operations to run in float16 while retaining float32 for numerically sensitive calculations. This lowered GPU memory usage without compromising model accuracy noticeably, and enabled the training of deeper models within the same memory limit.
All numeric columns in the dataset were downcasted using a custom function. Floats were reduced from 'float64' to 'float32', integers were cast to 'int32' or smaller, and object columns with low cardinality were converted to categorical. This reduced memory for the dataset by 40–45%, decreasing the overhead, without degrading performance.
To model performance in speed on the different LSTM setups. The approximate number of total parameters for each trial was calculated, and referenced in percent against a known configuration. This gave an indication of what time was expected for that configuration given it was on the reference setup, ignoring overhead. The peformance gain from not using GPU no downcasting, no mixed precision, cuDNN to utilizing all of these, resulted in a performance gain in speed from around 56 seconds per epoch to around 3 seconds, signifying the importance of the computational setup and correct data management. The next major future speed gains could most likely be found in reducing the overhead.

# B Supplementary Application

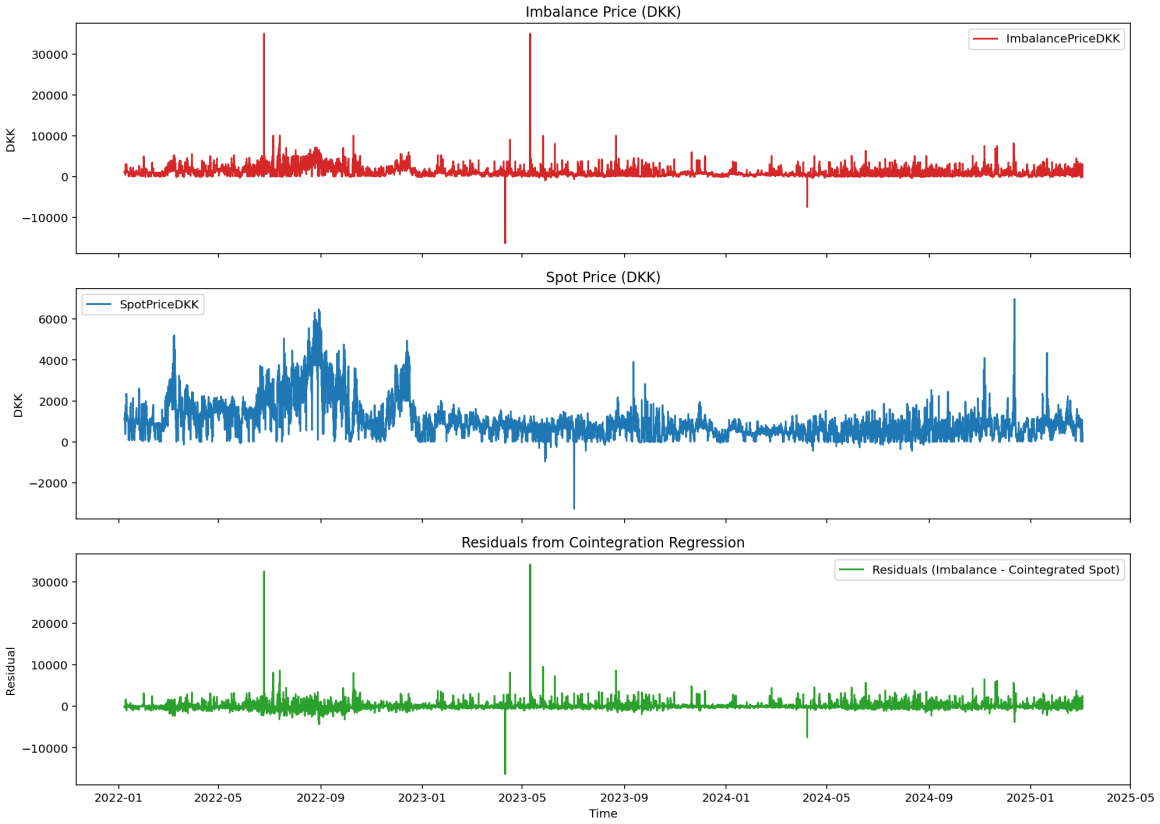


***Figure B.1:*** *DA Spot, Imbalance price, and co-integrated residuals confirming by visual inspection first order stationarity*

# References


Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794. ACM.

Frazier, P. I. (2018). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Hinton, G. (2012). Lecture 6e: Rmsprop—divide the gradient by a running average of its recent magnitude. https://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf. Lecture notes, Neural Networks for Machine Learning, Coursera.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. https://deeplearning.cs.cmu.edu/F23/document/readings/LSTM.pdf. Lecture notes, Carnegie Mellon University.

Hossain, M. A. and Rahman, M. M. (2023). A systematic literature review: Recursive feature elimination algorithms. *Scite*.

Keskar, N. S. and Socher, R. (2017). Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Liu, Y., Gao, Y., and Yin, W. (2020). An improved analysis of stochastic gradient descent with momentum. In *Advances in Neural Information Processing Systems*.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30, pages 4765–4774.

Ronderos, N. (2019). Seasonal unit root tests. https://blog.eviews.com/2019/04/seasonal-unit-root-tests.html. Accessed: 2025-05-27.

Schaeffer, R., Khona, M., Robertson, Z., Boopathy, A., Pistunova, K., Rocks, J. W., Fiete, I. R., and Koyejo, O. (2023). Double descent demystified: Identifying, interpreting ablating the sources of a deep learning puzzle.

Shavlik, J. (2015). Long short-term memory networks. https://pages.cs.wisc.edu/~shavlik/cs638/lectureNotes/Long%20Short-Term%20Memory%20Networks.pdf. Lecture notes, University of Wisconsin–Madison.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

Vergara, J. R. and Estévez, P. A. (2014). A review of feature selection methods based on mutual information. *Neural Computing and Applications*, 24:175–186.

Wang, D. and Tomek, W. G. (2004). Commodity prices and unit root tests. In *Proceedings of the NCR-134 Conference on Applied Commodity Price Analysis, Forecasting, and Market Risk Management*, St. Louis, Missouri.

Wikipedia contributors (2024). Shapley value — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Shapley_value. Accessed: 2025-05-26.

Zhou, Z. and Hooker, G. (2020). Unbiased measurement of feature importance in tree-based methods.