Bachelor thesis

Studying the Relationship between Window Size and Other Hyperparameters in Deep Learning-based Energy Consumption Forecasters

Elliot Johnson (434349)

Supervisors:
Prof. Dr. Holger H. Hoos
Dr. Mitra Baratchi (2nd Reader)
Wadie Skaf, M.Sc. (Daily Supervisor)

RWTH Aachen University
Department of Computer Science
Chair for AI Methodology (Informatik 14)
https://aim.rwth-aachen.de/

Communicated by Prof. Dr. Holger H. Hoos

Abstract

The forecasting of time series data has brought numerous benefits since its inception in statistics. Specifically, the energy sector has witnessed the deployment of Machine Learning (ML) algorithms. As electrical grids are dynamic systems that need to be balanced, forecasting their fluctuations in demand is important. However, the notorious difficulty with time series analysis lies in window size selection. While training ML models, engineers must select an uninterrupted window from which the model predicts the next value. This thesis investigated the relationship between the window size and other hyperparameters in deep learning-based forecasters. Subsequently, Bayesian Optimisation (BO) was implemented using novel surrogate models such as the combined surrogate model, which balances the strengths and weaknesses of a Random Forest (RF) and a Gaussian Process (GP). In some cases, this new combined model outperformed existing surrogate models. Additionally, conformal methods were integrated into the SMAC library, but their performance in optimising forecasters did not beat that of pre-existing methods. Using the optimisation data collected, an analysis of the importance of the window size and its influence on performance in combination with other hyperparameters was conducted.

Table of Contents

List of Figures

List of Tables

List of Symbols

$\hat{q}$	Conformal threshold used for prediction validity.
$\hat{t}_\gamma(x)$	Learned quantile regressor for quantile γ and input x.
$t_\gamma(x)$	True quantile regressor for quantile γ and input x.
$C(x)$	Conformal prediction set for a given input x.
$EI(\theta)$	Expected improvement of a configuration if the data is not log transformedθ.
$EI_{\exp}(\theta)$	Expected improvement of a configuration if the data is log transformedθ.
M	Set of the smallest possible partitions of the hyperparameter space according to a interaction metric threshold m.
N	Set of all hyperparameters.
P	Partition of hyperparameters in the configuration space.
U	Subset of the hyperparameters N.
X	Set of all inputs.
Y	Set of all labels or outputs.
Φ	Cumulative distribution function of a standard normal distribution.
α	Probability that the correct output is contained in a prediction set.
γ	Quantile.
$\mathbb{F}_U$	Importance or interaction effects of a set of hyperparameters.
$\mathbb{V}$	Variance of predicted performance metrics.
μ	Mean of a distribution or prediction.
$\mu_{\log}$	Mean of a log normal distribution.
ϕ	Probability density function of a standard normal distribution.
σ^2	Variance of a distribution or prediction.
$\sigma^2_{\log}$	Variance of a log normal distribution.

θ	Configuration of hyperparameters in a model.
f_{min}	Empirical mean performance of the incumbent configuration.
m	Threshold for interaction metric significance.
$s(x, y)$	Uncertainty measurement for input x and label y.
v	$\frac{\ln(f_{\min}) - \mu_\theta}{\sigma_\theta}$.
x	An individual input.
y	An individual label.
y_max	The largest value of a set of target feature values.
y_min	The smallest value of a set of target feature values.

Acronyms

AutoML	Automated Machine Learning.
BO	Bayesian Optimisation.
CNN	Convolutional Neural Network.
DL	Deep Learning.
DNN	Deep Neural Network.
eCDF	Empirical Cumulative Distribution Function.
fANOVA	Functional Analysis of Variance.
GNLLL	Gaussian Negative Log-Likelihood Loss.
GP	Gaussian Process.
GRU	Gated Recurrent Unit.
HPO	Hyperparameter Optimisation.
ICP	Inductive Conformal Prediction.
LSTM	Long Short-term Memory.
ML	Machine Learning.
NN	Neural Network.
QR	Quantile Regressor.
RF	Random Forest.
RMSE	Root Means Squared Error.
RNN	Recurrent Neural Network.

SMAC	Sequential Model-based Algorithm Configuration.
SMAPE	Symmetric Mean Absolute Percentage Error.
TCP	Transductive Conformal Prediction.

Chapter 1

Introduction

The demand for accurate forecasting models is apparent in many sectors, such as finance, to predict prices and aid logistical systems [CFS04; AD21] or in meteorology, to predict the weather [Pri+25]. Specifically, models have been implemented in the energy sector to forecast energy market prices [FM20] or the energy consumption of buildings [SMR21]. Forecasting energy consumption is a vital tool for reacting to energy demands, as it allows for the utilities to properly sustain electricity across a grid [SMR21]. Therefore, the ability to train reliable forecasting models is of significant real-world value.

Given the large amount of data generated by electrical grids [Zai+21], ML-based solutions are popular options for building such forecasters. Specifically, it has been shown that a Neural Network (NN) can achieve higher accuracy than popular linear models in forecasting tasks [STN18; SSR19]. An important decision when training forecasters is the selection of hyperparameters. This set of values, or configuration, defines aspects of the model structure and training process [Dia+17]. Among the hyperparameters for forecasters is the window size, which defines the number of historical data points the model receives to make a prediction. While the search space of configurations depends on the model type, the window size must always be chosen, with the decision significantly impacting model performance [FM20; IJR17]. Still, given the non-monotonic relationship between window size and model accuracy [SJN22], this selection problem is not trivial. Therefore, a thorough analysis of the effect of this hyperparameter and suggestions for a systematic selection are of great value.

The optimisation problem of hyperparameters for NNs is known to be very difficult [Dia+17]. Modern approaches use SMAC to optimise the configuration used [Den+22]. SMAC, which stands for Sequential Model-based Algorithm Configuration, offers an implementation of BO that can be combined with a selection of surrogate models. Traditionally, RF is chosen, as it natively supports categorical hyperparameters and allows for high-dimensional configuration spaces required for Hyperparameter Optimisation (HPO).

However, an RF can often be outperformed by a GP in numerical spaces with a low number of dimensions [Bis+23]. For this reason, the potential exists for a novel ensemble surrogate model for SMAC that combines RF with GP using an approach similar to model stacking. This balances the strengths and weaknesses of each surrogate model when considering different types of hyperparameters.

While ensemble learning has the potential to increase accuracy, another paradigm in ML offers methods to improve performance as well. Conformal methods in ML allow for statistically sound prediction ranges based on previously seen data [AB22]. Surrogate models represent a promising application of these methods, as the accuracy of both the prediction and the uncertainty estimation of the model are essential. Thus, conformal methods indicate a viable improvement for the optimisation process. The integration into SMAC includes the implementation of a Conformal Quantile Regressor (CQR) as a new surrogate model and independent Thompson sampling as an acquisition function based on the ideas detailed in Salinas et al. [Sal+23].

Chapter 2

Background and Related Work

2.1 Forecasting using Deep Learning

The application of NNs to time series forecasting has long been of interest, with early work dating back to 1987 [LF87]. The non-linear, data-driven, and self-adaptive nature of NNs is considered advantageous over traditional forecasting methods [ZPH98]. Following developments in NN-architectures [Liu+17; Vas+17], improvements in computer performance [Lei+20], and the broader recognition of the relevance of big data [Zai+21], much research has been conducted to engineer increasingly accurate forecasters based on Deep Learning (DL). An essential trend driving the increase in performance of new NN-architectures is the combination and development of new neural network structures [Zho+21; SMR21]. This has led to a diverse collection of forecasters that vary significantly in design.

Darts [Her+22] is a Python library that contains implementations of some of the most popular forecasting architectures, such as N-BEATS [Ore+20], Temporal Fusion Transformers [Lim+21], RNN, and other deep learning-based models. N-Beats and Temporal Fusion Transformers are modern architectures that underpin the viability of DL-based techniques for forecasting. RNN is an established architecture comprising a far simpler base architecture than many state-of-the-art forecasters [Als+22]. Similar variations have been thoroughly tested and established in the use case of forecasting [SSR19; STN18].

The relevancy of forecasting models is exemplified when considering their application in energy consumption prediction [AZY20]. The challenge of forecasting energy consumption is widely relevant and encompasses the trends found in both single buildings [SMR21; QAR24] and entire grids [Kau+22]. This is especially relevant to legislative measures that motivate a switch to renewable energy and more efficient energy management, which requires accurate forecasting to plan future loads properly [AZY20].

Somu, MR, and Ramamritham [SMR21] developed the deep learning architecture kCNN-LTSM. Their model combined k-means clustering, Convolutional Neural Network (CNN) and LSTM neural networks to predict the energy consumption of a building.

Zhou et al. [Zho+21] developed the Informer model, a transformer-based model that was found to excel at long sequence time-series forecasting and outperformed the other models tested. To evaluate their model, various time series datasets were used to test Informer, including the ET-Dataset [Zho+21] and ElectricityLoadDiagrams20112014 [Tri15]. In addition, they tested different window sizes and alluded to the complexity of the problem. Both performance and training time were affected, underpinning the importance of selecting the correct window size.

Quantifying the effect that different hyperparameters have on performance, such as the window size, can help make conclusions on how to select the best configuration. The state-of-the-art solution for such analysis is Functional Analysis of Variance (fANOVA) [HHL14], which quantifies importance and interactions. This importance signifies the effect of a single hyperparameter on the resulting performance of the model. Quantifying the interactions can indicate the strength of the underlying relationships between hyperparameters. An RF is trained on performance data of different configurations, and the model is used to quantify the importance of hyperparameters

and to evaluate their interactions. This is not done by sampling a model; instead, fANOVA uses the complete trained model to deduce such metrics. The metrics that can represent such effects are, in essence, the results of analysing the variance in predicted algorithm performance between configurations. Essentially, the variance of all predicted performance metrics $\mathbb{V}$ for all possible configurations. This variance can be split into the following components:

$$\mathbb{V} = \sum_{U \subset N} \mathbb{V}_U, \text{ where } N \text{ is the set of all hyperparameters}$$

$\mathbb{V}_\mathbb{U}$ is the measure for the average effect that setting the hyperparameters in U has [Hoo07]. The importance of a single hyperparameter, $|U| = 1$, or the interactions between multiple hyperparameters, $|U| > 1$, are then given by $\mathbb{F}_U = \mathbb{V}_U / \mathbb{V}$
Deng et al. [Den+22] used fANOVA to analyse the hyperparameter importance distribution of forecasting DNNs. They found that the importance was not evenly distributed with the window size amongst the most important, which coincided with other conclusions by Fezzi and Mosetti [FM20] and Inoue, Jin, and Rossi [IJR17].
Ermshaus, Schäfer, and Leser [ESL23] formulated a review of different window selection methods and compared their performance on different unsupervised analysis tasks. Although forecasting is not an unsupervised task, it is relevant to recognise that the selected windows outperformed the window sizes selected by humans in the realm of anomaly detection and segmentation. This indicated the potential to find automated techniques that outperform manual selection.
A more conventional approach of selection is the use of a global HPO to find values for hyperparameters that result in a well-performing model [Als+22]. This optimises the window size as well and ideally results in a well-selected value. One downside of this method is that all hyperparameters are treated equally, which does not necessarily reflect the distribution of impact that hyperparameters have on the error of the resulting model. However, it gains an advantage with the ability to take the interactions of different hyperparameters into account.

2.2 Bayesian Optimisation

BO has established itself as a leading optimisation approach for hyperparameters [Den+22; HHL11]. It sequentially tests new configurations and uses the experience to suggest new configurations to be trained with the goal of minimizing the cost metric of the trained configuration. The key advantage of BO over other search methods, such as random search, lies in the use of ML-models to model the resulting performance of a given configuration [HHL11]. A key instrument in this is the uncertainty quantification of the model in some regions. Given the many dimensions of the configuration space and the expensive nature of algorithm execution, the amount of data available is usually extremely limited. What follows is the risk that many regions of the space may remain unexplored, potentially losing out on promising configurations. This implies that a balance must be struck between exploration and exploiting the configurations of well-represented regions in which the model can more accurately find promising configurations. The key to this balance is the acquisition function, which delivers a value that represents a combined assessment of the potential gain of the configuration given the predicted cost and the corresponding uncertainty. The potential gain is estimated to be higher with high uncertainty as the model has not explored the region yet. This means an evaluation will improve the prediction capabilities of the model in that region. A configuration will also receive a high assessment if the predicted cost of the configuration is low. This means the configuration has potential as a new incumbent, which is the configuration with the best empirical performance.
Many tools exist that offer some variation of BO, but an established option is the Sequential Model-based Algorithm Configuration (SMAC) library [Lin+22; HHL11]. SMAC offers much customisation for the algorithm configuration process and is specifically intended for expensive target functions. These functions depend on a predefined set of hyperparameters and can entail any kind of algorithm that returns a cost metric to be minimized. This makes it ideal for the configuration of a Deep Neural Network (DNN) as the training process can demand many computer resources [FH19], and the error on a final validation set can be used as a function return. Budgets can be

introduced that allow for a variation in training "effort", using a hyperparameter such as epoch count. SMAC sets a budget for a given configuration before training and takes this budget into account during performance comparisons. The driving principle behind the library is the combination of intensifiers, which are racing algorithms combined with BO. Intensifiers offer support for different instances and budgets, while BO helps determine what configuration to test next.
Finally, essential to BO is the performance of the surrogate model used, as the process relies on a proper assessment of the benefit of a configuration. SMAC expects a mean and variance for every prediction, but BO can work with any prediction and uncertainty quantification as long as the acquisition function is correct. Good surrogate models do not only have a high prediction accuracy, but they are also capable of accurate uncertainty estimations. A further important criterion for the applicability of a model in BO is the type of data with which the model can work well. Hyperparameters can be numerical or categorical in nature, meaning that the specific configuration space can influence the input space of the model. In contrast, the output is consistently continuous, as the cost metrics usually are. Hyperparameters can often be dependent on each other. These conditional hyperparameters appear if the setting of a specific hyperparameter influences the existence or choices of another. An example of this is the learning rate scheduler hyperparameter, which may require further hyperparameters unique to the scheduler chosen. Due to the variable shape of the input for the model, it is important to consider the configuration space when deciding on a surrogate model.
As of writing this, two surrogate model implementations exist in SMAC besides a random model. GPs are an established component of BO for expensive black box functions [Lin+22]. GP works well with continuous inputs but is ideally adjusted to function properly with discrete inputs [GH20]. Furthermore, configuration spaces with more than 10 hyperparameters or conditional hyperparameters traditionally prove to be more difficult to support [Bis+23]. Due to these constraints, GP has not been the typical choice for the task of optimising hyperparameters but has proven to work well in numerical spaces [Egg+13].
The second of the two models available in SMAC is RF. It is capable of natively handling categorical data and conditional hyperparameters and can accept numerical values in its input space through dynamically finding thresholds. As configuration spaces can contain up to hundreds of hyperparameters, it is important to note that RF seems to scale relatively well to high-dimensional spaces with mixed hyperparameters [Egg+13]. For this reason, RFs have been the surrogate model of choice when optimising hyperparameters. However, promising work has been done to challenge this convention using various machine learning techniques such as tree-structured Parzen estimators [Ber+11] or Conformal Quantile Regressors (CQRs) [Sal+23].

2.2.1 Ensemble Models

Ensemble learning is a paradigm in ML that aims to combine machine learning models to increase performance [RG20]. This goal is derived from the inherent understanding that models are imperfect. The intuition depends on the assumption that a collection of models will lead to a compensation of such imperfections and increase overall accuracy. Since the use of surrogate models based on ML has gained prominence in the realm of optimisation processes, the use of ensemble techniques has also been found to enhance the performance of models [Goe+07]. Surrogate models are intended to reflect the relationship between configuration and performance. The accuracy of this reflection impacts the overall performance of the optimisation. However, when specifying which surrogate model to use, there may be many options to choose from, each with strengths and weaknesses.
Combining multiple models can be executed in many different ways. One such method is called model stacking, in which each model is allowed to make an independent prediction based on the input data. After each model in the ensemble has produced a result, the outputs are combined by a meta-learner to produce the final output visualised in Figure 2.1. In this case, the meta learner only takes the outputs of the models to combine towards a general output that is accepted. It is generally accepted that the training data set must be split for this process, such that the training data used for the meta learner has not been seen by the base learners [Wol92].

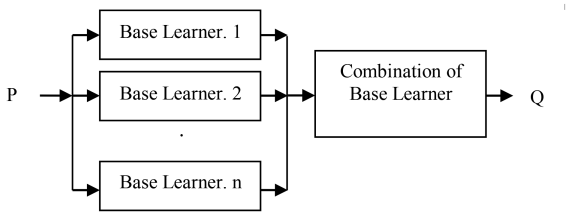


Figure 2.1: A Basic architecture of Model Stacking [RG20]

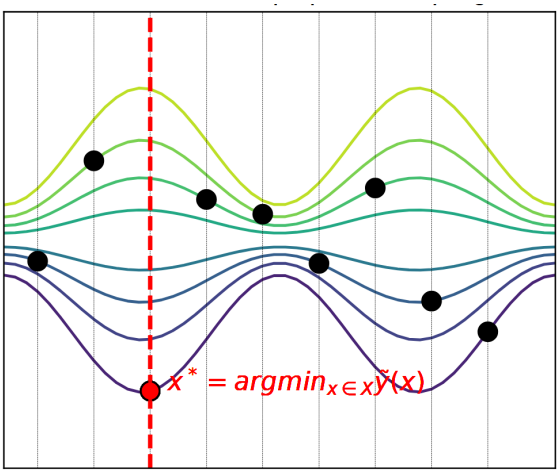


Figure 2.2: Visualisation of eight quantile regressors [Sal+23]

The method implemented for surrogate models by Goel et al. [Goe+07] aggregates the predictions of multiple models with an assigned weight. Firstly, the performance of each model is evaluated through cross-validation. The average error of each model is then used to determine weights for the models. The weights collectively sum to one and represent the influence of the respective model on the resulting prediction. This resulted in an overall increase in performance in comparison to simpler methods, such as choosing the single best-performing model according to cross-validation.

2.2.2 Conformal Prediction

A major challenge in machine learning is the inherent imperfection of the predictions. For this reason, models can often include a measure of uncertainty given a prediction, such as the range described by multiple quantile regressors as can be seen in Figure 2.2 or the variance of an RF. These measures can be converted to scores of uncertainty, denoted as $s(x, y)$ for an input x and label y, where high scores represent a high uncertainty. However, these are merely based on an assessment from the model itself, thus, they can not be completely trusted. Conformal prediction offers a method to externally determine statistically sound prediction sets and ranges that include the true value with an arbitrarily high probability [AB22]. As with any ML process, the data is the leading resource for conformal prediction with two approaches regarding its handling. Transductive Conformal Prediction (TCP) utilises the complete data set for training but greatly increases computational complexity, as the model has to be retrained as often as there are training instances. On the other hand, Inductive Conformal Prediction (ICP) is much less expensive but requires the data to be split into a training and calibration set. For this reason, it is only used when enough data is available. ICP can be applied to a variety of prediction tasks.
For classification problems, a conformal prediction set $C(x)$ for an input x must contain the true class with an arbitrary probability of at least $1 - \alpha$ where $\alpha \in [0, 1]$. In a nutshell, conformal methods use the uncertainty assessments $s(X_c, Y_c)$ for the calibration input X_c and output Y_c data to deduce a threshold $\hat{q}$ such that, with a probability of $1 - \alpha$, the true label y_c of a sample x_c

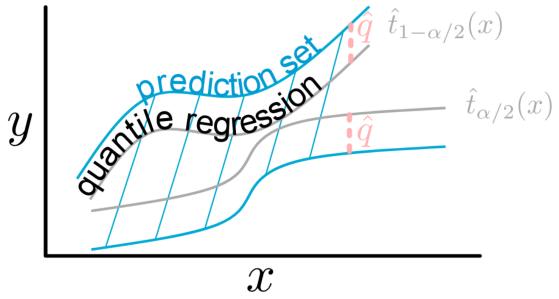


Figure 2.3: Visualisation of a pair of conformal quantile regressor [AB22]

receives an uncertainty metric $s(x_c, y_c) \leq \hat{q}$ from the model as can be seen in Equation 2.1. As long as the calibration is representative of the true distribution, this ensures that the final prediction sets contain the true label with a high probability.

$$C(x) = \{y \mid s(x, y) \leq \hat{q}\} \tag{2.1}$$

Prediction intervals for numeric outputs that contain the true value with the same arbitrary probability of $1 - \alpha$ can be achieved with the help of quantile regressors, as can be seen in Figure 2.3. Quantile regressors $\hat{t}_\gamma(x)$ model the true γ quantile $t_\gamma(x)$ where $\gamma \in [0, 1]$. To represent our desired arbitrary interval we write $\hat{t}_{\alpha/2}(x)$ and $\hat{t}_{1-\alpha/2}(x)$. These can be trained using any model with a quantile loss function, as can be seen in Equation 2.2.

$$L_\gamma(\hat{t}_\gamma, y) = (y - \hat{t}_\gamma)\gamma \mathbb{1}\{y > \hat{t}_\gamma\} + (\hat{t}_\gamma - y)(1 - \gamma)\mathbb{1}\{y \leq \hat{t}_\gamma\} \tag{2.2}$$

However, a similar problem with conformity holds as the resulting interval only depends on the imperfect, predicted bounds. To train Conformal Quantile Regressors (CQRs), an offset $\hat{q}$ is calculated instead of a threshold. Samples from the unseen calibration set are used to compare predicted intervals to true values Y_c given inputs X_c. Then, $\hat{q}$ is adjusted such that, with a probability of $1 - \alpha$, given an input x_c and true output y_c the following holds $y_c \geq \hat{t}_{\alpha/2}(x_c) - \hat{q}$ and $y_c \leq \hat{t}_{1-\alpha/2}(x_c) + \hat{q}$ as can be seen in Equation 2.3.

$$C(x) = [\hat{t}_{\alpha/2}(x) - \hat{q}, \hat{t}_{1-\alpha/2}(x) + \hat{q}] \tag{2.3}$$

The regression problem of model-based HPO offers an interesting use case for Conformal Quantile Regressors (CQRs) due to the statistically sound uncertainty estimations. HPO heavily relies upon the proper estimation of regressor uncertainty, implying a potential improvement in performance. Salinas et. al. [Sal+23] have found success when using Thompson sampling with Conformal Quantile Regressors (CQRs).
Thompson sampling is a heuristic used to balance exploration and exploitation that has experienced an increase in recent interest [CL11] despite its early inception in [Tho33]. Commonly, the goal is defined as maximizing the total reward of a series of actions. In hyperparameter tuning, an action may be a function evaluation, and a high reward would be a low cost. While the goal of HPO differs slightly as only the reward of the best action is judged, Thompson sampling has still seen application in BO [Kan+18]. In essence, a probability distribution is built for every action that represents the probability distribution of the reward [CL11]. This distribution is then sampled for every action, and the action with the best reward is chosen.
To employ this principle practically, multiple quantile regressors can represent a distribution of the possible rewards per action, as can be seen in Figure 2.2. For instances with too many possible actions, such as in HPO, a random collection of actions is taken. To recreate the effect of randomly sampling the distribution of each action, a quantile represented by the regressors is chosen at random for each action. Finally, the action with the highest reward is chosen to be evaluated.

Chapter 3

Objective and Approach

The objective of this research was to gain insights into the window size hyperparameter given deep learning-based models trained to forecast energy consumption data. Specifically, we aimed to answer the following research questions:

1. How important is the window size regarding building deep learning-based time series forecasters?
2. What is the relationship between window size and other hyperparameters, and can any trends or patterns be found with the performance of the model?
3. Does the quality of optimisation increase when the relationship between the window size and other hyperparameters is considered?

To make the first observations, data had to be gathered regarding the performance of specific configurations as a first step. This meant using optimisation runs to make conclusions about the importance and interactions in the configuration space of the selected models. After implementing the context in which forecasters could be trained on data and tested, they were optimised using pre-existing techniques. Using preliminary testing, conclusions were made about how the window size interacted with other hyperparameters. This offered insight into the hyperparameter and inspired methods in which some hyperparameters would be optimised semi-separately.
Given the new findings, we explored novel approaches with the potential of improving the optimisation performance of the well-known SMAC library. These surrogate models required extensive configuration after their basic implementation to determine the ideal settings. To determine which configurations work well, tests for the accuracy of the surrogate model and its performance during actual optimisation were conducted. This included implementing test infrastructure that ensured fair testing conditions, such as minimizing differences due to random initial configurations by injecting the same initial configurations for all instances. Synthetic functions were also used to minimize the computational cost of expensive deep learning runs to the final tests after tweaking the new model was complete. The final tests of optimising deep learning forecasters used the same testing infrastructure but were not repeated as often due to cost. Finally, using the data gathered from testing, a final analysis of the hyperparameter patterns and relationships was conducted. This included updating previously calculated importance and interaction metrics. Due to the increase in available data, the new metrics more accurately reflected reality.

Chapter 4

Methodology

4.1 Environment and tools

The nature of studying and optimising hyperparameters implies the existence of some context. This context involved a model whose hyperparameters were to be optimised and the data on which the model was trained, validated, and tested. While training forecasters on energy data was at the forefront, synthetic data was used to perform preliminary tests on the different implemented HPO methods. Using cost-efficient functions to evaluate the performance of many different optimisers is much more reasonable due to the computationally expensive nature of deep learning. Garrido-Merchán and Hernández-Lobato [GH20] evaluated using synthetic functions to assess the performance of their new surrogate model. The Rosenbrock function, as shown in Equation 4.1, is a synthetic function used to test different black-box optimisation algorithms [Elh+19].

$$f_{ros}(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \right] \tag{4.1}$$

This function has been applied in benchmarking HPO [Che+22; Tan+21] due to its flexible dimension count and complex topography with multiple local minima if there are more than three dimensions [SQ06].
After testing, a forecaster was to be chosen which would be optimised by the developed methods. A wide variety of deep learning-based models exist with state-of-the-art ideas and modifications that aim to raise the ceiling of achievable performance [Zho+21; SMR21]. In addition to this, the much more rudimentary models such as RNN and its derivatives LSTM and Gated Recurrent Unit (GRU) were chosen. These are much more established architectures with independent confirmations of their superiority over simpler methods [STN18; SSR19]. Furthermore, the simple implementation in Darts [Her+22] with support for custom PyTorch optimisers and learning rate schedulers made this a reasonable choice. It is important to keep in mind that the importance of the absolute final performance of the trained models comes second to the relative performances between configurations used to compare optimisation processes.
To study the effects of different hyperparameters on the performance of a model using examples of configuration performances, the library fANOVA was used. It natively supports SMAC, allowing for a simple run analysis using previously generated data on various configurations. A preliminary analysis was conducted using a relatively small number of examples from which general conclusions about the importance and interactions that affect the different hyperparameters can be obtained. This enabled the influence of the window size to be confirmed early on and provided insights that led to further conclusions. Later, the analysis was expanded by utilising the many runs conducted for testing to confirm the results and represent the ground truth more. The findings led to the following hypothesis:

Hypothesis 1 (H1): *Subdividing the configuration space of an energy consumption forecaster according to interaction metrics can lead to a set of hyperparameters that have a relatively independent influence on performance.*

4.2 Partitioning the Configuration Space

The goal of partitioning the hyperparameters was to find partitions that have relatively few independent interactions. A threshold m was set to classify an interaction as significant. The conditions for a valid partition P of the set of all hyperparameters N is formalised using the effects $\mathbb{F}_U$ of a hyperparameter subset U. The definition can be found in the Equivalency 4.2.

$$P \text{ is a valid partition } \Leftrightarrow \; P \subseteq N \text{ and for all } h_i \in P \text{ there is no } h_j \notin P \text{ with } \mathbb{F}_{\{h_i,h_j\}} > m \tag{4.2}$$

As can be seen in the definition above, if P_i and P_j are valid partitions, so is $P_i \cup P_j$. These partitions can contain a variety of hyperparameters and have no major interaction effects with other hyperparameters outside.
Another feature proven useful is the maximisation of the total importance of the hyperparameters contained in a partition of a given size. First, the set M of the smallest possible partitions is required, as seen in Equation 4.3.

$$M := \{P | P \text{ is a valid partition and no subset of } P \text{ is a valid partition}\} \tag{4.3}$$

The total importance of a partition can be assigned as P as $\sum_{h \in P} \mathbb{F}_{\{h\}}$. The final goal is to find a set of these partitions to combine into one with a high total importance according to the individual hyperparameters. Trivially, the set can be constructed by considering all combinations that exceed the size limit and determining the highest total importance. Alternatively, an optimisation problem can be constructed with the importance as an objective function to be maximised and the number of hyperparameters as a constraint.

4.3 Ensemble Techniques

The leading motivation for employing ensemble techniques in the optimisation process comes from the following hypothesis:

Hypothesis 2 (H2): *Subdividing the configuration space and its optimisation according to interaction metrics can result in favourable results.*

The novel optimisation method is the result of considering the limitations of the surrogate models used in SMAC. The GP can handle small, continuous spaces with ease but struggles with handling conditional hyperparameters, large configuration spaces, and discrete spaces. While RF excels where the other struggles, it is less effective than GP in numerical spaces. The solution is to subdivide the configuration space and train the surrogate models on subdivisions that best suit their strengths. However, BO expects the output of only a single surrogate model to be used with the acquisition function. This indicates that an ensemble approach could be beneficial for mixing the output of multiple models. Specifically, an ensemble approach similar to model stacking was used as follows: Two models are designated as either a broad or supplementary model that are combined to form the surrogate model. The broad model processes all or most of the configuration space and produces a prediction and variance according to its training data. The training data consists of the configuration and empirical performance pairs continuously gathered during optimisation. However, the supplementary model only works with a smaller portion of the complete configuration space, depending on the training data. When SMAC begins a new optimisation process, an initial collection of configurations is tested. This provides the surrogate model with minimal data to suggest future configurations. We use these initial configurations to conclude the important hyperparameters and their interactions using fANOVA. The configuration space is then partitioned using the interaction metrics and important metrics, and an interaction threshold is derived from the distribution of interaction values. The supplementary model then supplements the broad model using the output of the latter and the partitioned input data to make a final prediction. This supplementary prediction is the prediction used by the acquisition function. The internal

processes between the different models inside the combined model can be seen in Figure 4.1 and in Figure 4.2

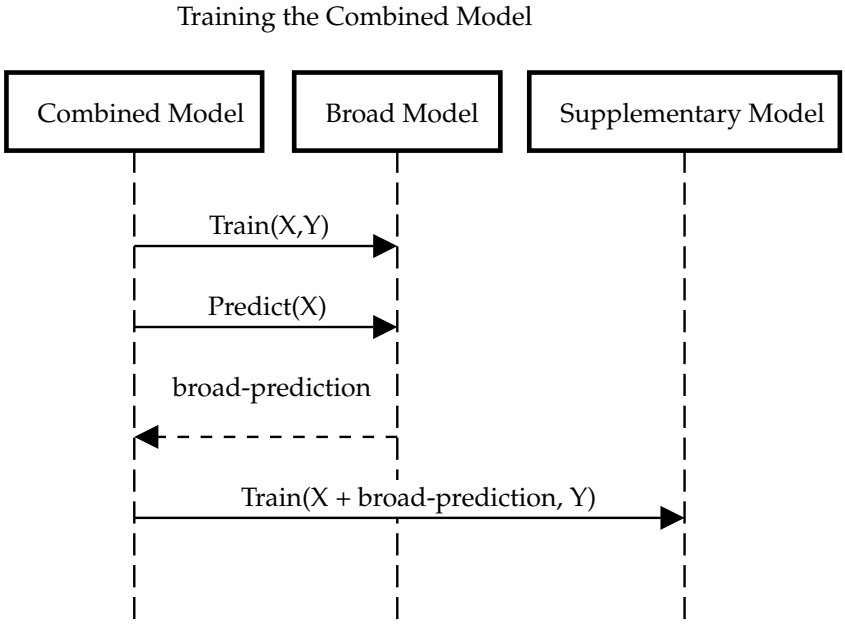


Figure 4.1: Visualisation of a training process for the combined model

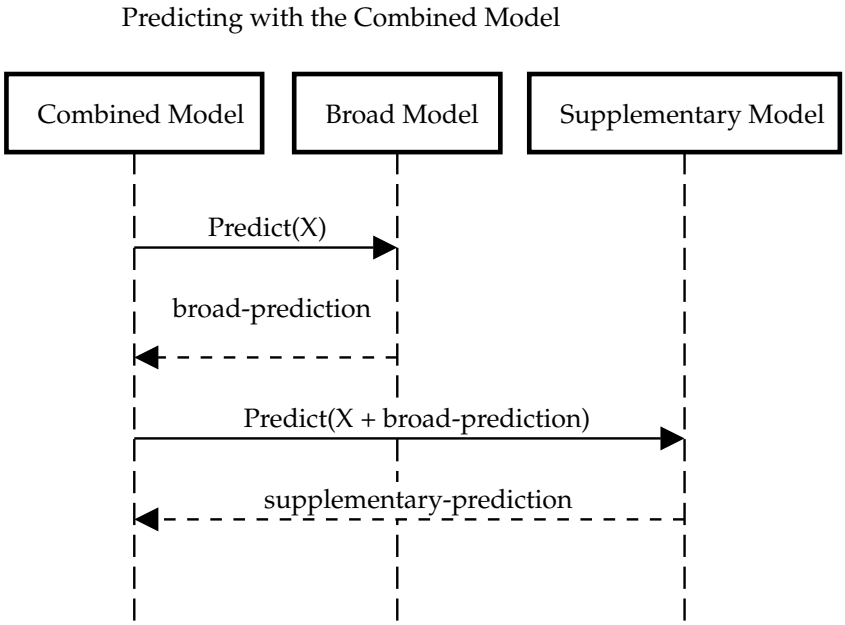


Figure 4.2: Visualisation of a prediction process for the combined model

Many versions were composed, consisting of specific setting variations. These settings can be summarised as affecting different parts of the supplementation process.
The order of model training had a small influence on the potential of the supplementary model to overfit to the output of the broad model. If the broad model has seen the complete training data, its predictions on this training data will not be representative of its performance on unseen data.
If the supplementary model is trained on an older version of the broad model, it may be less susceptible to overfitting because the latter has not seen newer portions of the collected data. This

difference in training data amount becomes more negligible as the optimisation progresses because surrogate models are trained very frequently, meaning the amount of new data is consistently small. However, the supplementary model might underperform in correcting for the errors of the broad model because it does not train on the predictions of the model that will later be used for unseen estimations.
If supplementation is not capable of outperforming the broad model by itself in all cases and both seem to be advantageous in different scenarios, mixing the two can be promising. The method from Goel et al. [Goe+07] mixed multiple surrogate models using cross-validation to deduce weights that result in a combined final prediction.
Adjusting the broad model configuration space has an impact on the entire system. Although a reduction in space would imply a negative impact on average broad model accuracy, this may help the supplementary model. This might reduce the risk of overfitting because the ability to generalise is impacted, whereas the supplementary model still receives a performance estimate based on the configuration of unseen hyperparameters.
Splitting data is a very popular ensemble technique because it addresses the overfitting issue by gathering broad model predictions on unseen data. The downside of this method is that the final model trains only on a portion of the data. This means that when splitting, a balance must be established between the accuracy of the broad model and the supplementary model. The performance gain of avoiding overfitting may or may not outweigh the loss caused by a reduced instance number that is already small due to the expensive nature of gathering training data.
Finally, encoding the training data impacts performance. Surrogate models have specific limitations regarding different encodings. The SMAC class `HyperparameterOptimizationFacade` uses a log scale encoding, which also uses an RF as the surrogate model. This scales the performance metrics to the interval $[0, 1]$ and takes the logarithm, which has been shown to improve quality [HHL10]. When a GP is used during black-box optimisation, no change to the performance metrics is done. This is because the mean calculation is affected when multiple cost metrics for a single configuration are recorded. For the use case of hyperparameter optimisation, multiple cost metrics are handled differently because each has a corresponding budget. This means that multiple encodings are possible for a GP. The type of encoding affects what acquisition function can be used. The acquisition function used by SMAC in an HPO context is called Expected Improvement. Two versions exist that depend on the transformation of the configuration performance data. If the metrics experience a log transform when optimising, the function can be described with Equation 4.4, where θ represents an arbitrary configuration.

$$EI_{\exp}(\theta) := E[I_{\exp}(\theta)] = f_{\min}\Phi(v) - e^{\frac{1}{2}\sigma_\theta^2+\mu_\theta} \cdot \Phi(v - \sigma_\theta) \tag{4.4}$$

[HHL11]
If the performance metric does not undergo a log transformation before use, the expected improvement function is described by Equation 4.5.

$$EI(\theta) := E[I(\mathbf{x})] = (f_{\min} - \mu_\theta)\Phi\left(\frac{f_{\min} - \mu_\theta}{\sigma_\theta}\right) + \sigma_\theta\phi\left(\frac{f_{\min} - \mu_\theta}{\sigma_\theta}\right). \tag{4.5}$$

[JSW98]
Furthermore, it is important to test whether transforming the prediction from the broad model before passing it to the supplementary model is important. This is done when the RF is trained on log-transformed metrics and the GP is not. In this case, a transformation of the prediction may be required. As the broad model predicts a mean and variance of a normal distribution and our data has been previously log-transformed, the distribution of the true metric is of a log-normal distribution [JKB94]. The mean $\mu_{\log}$and variance $\sigma^2_{\log}$ of this log-normal distribution can be deduced using the mean μ and variance σ^2 of the normal distribution as seen in Equation 4.6 and Equation 4.7.

$$\mu_{\log} = \exp\left(\mu + \frac{\sigma^2}{2}\right) \tag{4.6}$$

$$\sigma^2_{\log} = \left[\exp\left(\sigma^2\right) - 1\right]\exp\left(2\mu + \sigma^2\right) \tag{4.7}$$

These adjustments describe various possible settings for the final combined HPO method. Different combinations were compared to one another to deduce the best settings. As some settings were only later realised, settings were at times altered sequentially. This means that to test new settings, only the best-performing version at the time was considered to evaluate which of the new settings worked best.

4.4 Window Size Behaviour

The window size behaviour regarding deep-learning forecasters was analysed using various methods. Most importantly, any significant conclusions were made using large amounts of configuration data. The configuration data was obtained from testing various optimisation methods. Configuration performances from RNNs and LSTMs were collected using two datasets from the ETDataset: ETTh1 and ETTm1. The importance and interaction metrics were generated using fANOVA. Importance metrics were used to assess the importance of the window size hyperparameter for our use case. At the same time, the interaction matrix provided insights into which hyperparameters were most likely to experience a higher number of interactions.
Finally, scatter plots were used to plot the influence of the window size hyperparameters. Both 2D and 2D scatter plots with colour were chosen to display different types of influences. The 2D scatter plot shows whether a direct relationship between performance and window size is recognisable. This allowed for a conclusion to be made regarding dependency on other hyperparameters. To obtain specific information about possible trends, 2D scatter plots with hues that represented the cost were used. These were generated for hyperparameters with a high interaction metric with the window size.

Chapter 5

Implementation

5.1 Deep Learning Forecasters

The model implementations used in this research are from the Darts library [Her+22], which uses PyTorch Lightning to implement the deep-learning forecasters contained. The RNN implementation, along with its variations LSTM and GRU, were selected to be integrated into a new forecaster class. This class includes the configuration space for the forecasting model, the data module containing all important data, and a target function used by SMAC.
The configuration space was derived by taking the parameters that the RNN implementation uses and transferring it to a configuration space object of the ConfigSpace Python library [Lin+19]. Darts defines one RNN model class and sets the variation with a hyperparameter called "model_type". To include all three variations as separate forecaster classes, the same configuration space was modified to include only the corresponding model type. Many additional PyTorch hyperparameters were added, affecting the optimiser and learning rate scheduler. In addition, each possible learning rate scheduler received hyperparameters representing its parameters, which brought the total number of hyperparameters to 35. Finally, conditions and forbidden clauses were added to map hyperparameter dependencies and value incompatibilities[1].
Each dataset is managed by data module objects that run a setup function, splitting the data into training, validation, and testing sets. Following the work of Zhou et al. [Zho+21], the splits were 12/4/4 months for training, validation, and testing, respectively. The data module is a property of the forecaster class, which is used in the target function.
The target function is later used by SMAC to test a new configuration. This means the target function converts a set of hyperparameter values to a performance metric belonging to this configuration. Firstly, the hyperparameter values are converted to a format that the Darts class accepts. The initialised model is then fit to the training data, with the validation set being used for functionalities such as early stopping. Early stopping was chosen to improve training time by stopping the process if the increase in performance was negligible. Finally, the model is evaluated using the test data. The model receives the training data appended by the validation set and has to predict the entire test set. Using the prediction and true values, the SMAPE [HA18] is calculated and returned as a performance metric.

5.2 Combined Surrogate Model

SMAC includes an inherited abstract model to allow all other integrations into SMAC to function properly. A hierarchical class structure was chosen to implement the combined surrogate model. Due to the many versions implemented, a hierarchical structure allowed for an efficient implementation of new ideas[2].

[1] See Table A.2 in the Appendix for the full list of hyperparameters.
[2] See Figure A.1 in the Appendix for a visualisation of the class hierarchy

The parent of all combined models is the Abstract Combined Model, which implements the basic ideas behind calculating the configuration space of the supplementary model. This is done after the initial configurations are gathered and the optimiser first trains the surrogate model. To allow for a dynamic recalculation of the supplementary configuration space, the configuration count is tracked by the model, enabling the user to define at what points in the optimisation to update the configuration space. By default, one singular space calculation is executed the first time the surrogate model is trained. The training data is then sent to a wrapper around fANOVA that preprocesses the training data. To fulfil compatibility requirements, a new configuration space is generated that only includes the hyperparameters that are always covered in the configurations of the training data. This means removing any hyperparameters that are conditional on others and thus sometimes contain a NaN value. Furthermore, since our downstream implementation includes a GP to supplement the broad model, we reduce the configuration space to only contain numerical hyperparameters. Thus, the calculations of importance and interaction metrics only include unconditional, numerical hyperparameters.
Partitioning the hyperparameter space requires grouping hyperparameters by significant interactions. These significant interactions are recognised using a simple approach to anomaly detection: a z-score threshold seen in Equation 5.1 where j is an interaction.

$$z = \frac{j - \mu_{\text{interaction}}}{\sigma_{\text{interaction}}} \tag{5.1}$$

Treating significance as outliers allowed us to ensure that the supplementary model works with a practical configuration space size. An approach that relies on a data-independent threshold risks categorising too many significant interactions, leading to large minimal partitions. If these minimal partitions are too large, the configuration space size constraint can hinder the supplementary model from accepting enough important hyperparameters. For this reason, a z-score threshold was chosen and calibrated such that reasonable minimal partition sizes were consistently being formed. After building the set of minimal partitions M, they are grouped to maximize the sum of importance. This grouping is done by comparing all possible combinations. This can be formulated and solved as an optimisation problem, but a brute-force approach was chosen for its simplicity. Due to the low amount of unconditional, numerical hyperparameters of the models tested, the expensive runtime of this approach is negligible. The final supplementary configuration space is then generated, which allows for the entire supplementary model to be initialised. As previously discussed, some versions initialise the broad model with a partition of the configuration space as well. For this reason, only once the hyperparameters of the supplementary model have been chosen can some versions set a broad model. The principle behind the ensemble idea requires all hyperparameters unseen by the supplementary model to be covered by the broad model. This creates an issue if hyperparameters are conditional on ones in the other space. To create a consistent configuration space for the broad model, some hyperparameters from the other model are included to satisfy any conditional statements. We called the broad configuration space mostly disjoint in this case. Table 5.1 shows the implemented versions for which this is the case.
Once the models have been initialised, training and predictions are possible. These implementations are included in the child models of the abstract combined model. Many implementations train the broad model before the supplementary model. This is important because the supplementary model requires a prediction from the broad model to be trained. If the model is trained after the supplementary model, the order must be reversed if the configuration space has not been calculated yet. After all, the broad model can not predict before being initialised. Data splitting is implemented by the model version DMS-v4. In this case, the training data examples are randomly split in half. The broad model is trained on one half while the supplementary model is trained on the second half. Training the broad model precedes generating predictions on the unseen half for the supplementary model.
Different methods to produce a final prediction introduced the most variability among versions. Different settings define if or how the output of the broad model is transformed. Furthermore, the option of calculating a weighted aggregation of the model outputs was implemented.

Version	Encoding	Loss Function Broad Space	Broad Output Transform Training Order
MS-v1	log Encoding	N/A	No Change
		Complete Coverage	Supplementary first
MS-v1.5	log Encoding	N/A	No Change
		Complete Coverage	Broad first
MS-v2	log Encoding	N/A	Range Transform
		Complete Coverage	Broad first
MS-v3	log Encoding	N/A	No Variance Transform
		Complete Coverage	Broad first
DMS-v1	log Encoding	N/A	No Change
		Mostly Disjoint	Supplementary first
DMS-v1.5	log Encoding	N/A	No Change
		Mostly Disjoint	Broad first
DMS-v1.5.1	Mixed Encoding	N/A	log-Normal Transform 1
		Mostly Disjoint	Broad first
DMS-v1.5.2	Mixed Encoding	N/A	log-Normal Transform 2
		Mostly Disjoint	Broad first
DMS-v4	log Encoding	N/A	No Change
		Mostly Disjoint	Broad first
WA-v1	log Encoding	RMSE	No Change
		Complete Coverage	Supplementary first
WA-v1.1	log Encoding	GausianNLL Loss	No Change
		Complete Coverage	Supplementary first
DWA-v.1.5	log Encoding	RMSE	No Change
		Mostly Disjoint	Broad first

Table 5.1: Table of the relevant combined model versions and the distinguishing settings of each one

The first transformations of broad model outputs tested if the format of the output was important for the performance of the model. Three options were tested for the supplementary input given a mean μ_{broad} and variance σ^2_{broad}:

- No Transform: $[\mu_{\text{broad}}, \sigma^2_{\text{broad}}]$
- Range Transform: $[\mu_{\text{broad}} + \sigma^2_{\text{broad}}, \mu_{\text{broad}} - \sigma^2_{\text{broad}}]$
- No Variance Transform: μ_{broad}

These rudimentary transformations tested if the format of the broad output has any major effect on performance. Once training data encodings were explored, a second reason for transformations became apparent. If the models use different data encodings, transforming the output from the broad model to fit the encoding of the supplementary model may increase accuracy. To achieve different encodings, each model is assigned a run history encoder. This encodes the target feature of the training data before training. The values stored in the broad model encoder are then used to reverse the log scaling. The formula for the mean and variance of a log-normal distribution, as seen in Equation 4.6 and Equation 4.7, was used to derive a reversal calculation of the scaling, which resulted in the following transformations:

- log-Normal Transform 1: $[\left(\exp\left(\mu_{\text{broad}} + \frac{\sigma^2_{\text{broad}}}{2}\right) \cdot (y_{\max} - y_{\min})\right) + y_{\min},$
  $\left(\left(\exp\left(\sigma^2_{\text{broad}}\right) - 1\right) \cdot \left(\exp\left(\mu_{\text{broad}} + \frac{\sigma^2_{\text{broad}}}{2}\right)^2\right) \cdot (y_{\max} - y_{\min})^2\right)]$

- log-Normal Transform 2: $[\left(\exp\left(\mu_{\text{broad}} + \frac{\sigma^2_{\text{broad}}}{2}\right) \cdot (y_{\max} - y_{\min})\right) + y_{\min}, \sqrt{\left((\exp(\sigma^2_{\text{broad}}) - 1) \cdot \left(\exp\left(\mu_{\text{broad}} + \frac{\sigma^2_{\text{broad}}}{2}\right)^2\right) \cdot (y_{\max} - y_{\min})^2\right)}]$

The $y_{\min}$ and $y_{\max}$ refer to the minimum and maximum performance values of the current collection of training examples. These are gathered from the encoder object and are slightly modified as done during the log-scale encoding process.
For both training and prediction, the supplementary model only receives the transformed output of the broad model. Finally, a version was implemented that includes a weighted aggregation of the two predictions. This was done according to the paper by Goel et al. [Goe+07]. The weights of each model are gathered at every configuration space calculation. After the models have been set, they are tested using cross-validation. To keep runtime low, leave p out validation was chosen instead of leave one out. Given parameters $\alpha < 1$ and $\beta < 0$, the average errors of the supplementary model E_{supp}, broad model E_{broad}, and both models E_{avg} are used to calculate weights as seen in Equation 5.2.

$$w^*_{\text{model}} = (E_{\text{model}} + \alpha \cdot E_{\text{avg}})^{\beta}, \qquad w_{\text{model}} = \frac{w^*_{\text{model}}}{\sum_{\text{model}} w^*_{\text{model}}} \tag{5.2}$$

The parameters were set as constants $\alpha = 0.05$ and $\beta = -1$ as per Goel et al. [Goe+07]. Once weight calculation was implemented, a representative error function was to be selected. The paper used RMSE as an error function, which was included as an option for the cross-validation calculation. However, RMSE only accounts for the predicted mean and doesn't take the uncertainty estimation of the surrogate model into account. For this reason, GNLLL was added for cross-validation, which takes both mean μ and variance σ^2 seen in Equation 5.3.

$$\mathcal{L} = \frac{1}{2}(\log(2\pi\sigma^2) + \frac{(y-\mu)^2}{\sigma^2}) \tag{5.3}$$

It allows for a proper evaluation of the entire predicted normal distribution. However, this loss function can result in negative values, which are not compatible with the weight calculations that depend on the errors being larger than 0. The solution was to set a new lower threshold and implement a safety measure if that lower threshold was crossed. Through testing, we found that the model rarely had an average prediction below -1, so we set -1 as the new lower bound to be subtracted from all errors. If the lowest average error of one of the models is lower than this, that value becomes the new threshold, which is subtracted from both errors. After calculation, the model weights are applied during every prediction. As the weight calculation is connected with the configuration space calculation, one can set the model to recalculate these weights once a sample count milestone has been reached. The versions that use weighted aggregation are designated with "WA" in their name.
Finally, a control for the combined model was included that tested the performance of the supplementary model alone. This was done by feeding the supplementary model with zeroes in place of the output of the broad model. This allowed the stand-alone performance of the supplementary model to be tested.

5.3 Conformal Quantile Regressors (CQRs)

CQR was the second surrogate model added to the SMAC library. The integration into the library was based on the work done by Salinas et al. [Sal+23]. The model allows one to define how many quantile regressors are to be used with a default of 10 for the conformal HPO method. Besides an extra quantile regressor that simply predicts a mean, all quantiles are evenly distributed in $(0, 1)$. During training the configuration data is split in half to build a training and calibration set, used for all internal regressors. Fitting and calibration are always done when the surrogate model is trained.

Predictions were implemented slightly unconventionally for SMAC. When predicting, a single quantile must be specified, which dictates from which regressor the prediction will originate. This is in contrast to the surrogate models that take no extra arguments besides the input configurations and output means and variances. For this reason, some slight changes to function definitions were required, along with a new implementation of the Thompson Sampling acquisition function. When the acquisition function calls for a prediction, it chooses a quantile regressor at random, besides the mean predicting regressor, and returns the prediction of that regressor for a given configuration. Outside of the acquisition function, SMAC calls upon the surrogate model when looking for the previously tested configuration with the best predicted mean. This is done in preparation for the maximisation process, so the conformal surrogate model must support this behaviour. The extra regressor contained in the list of models is the solution to this problem.
The quantile regressors chosen were RFs trained using the quantile loss function. This was done as the RF has been an established surrogate model for HPO. For this reason, the components of the SMAC HPO facade were kept, including the log scale encoding.

5.4 Testing Infrastructure

Both direct surrogate model tests and optimisation tests were implemented. Before optimisation tests evaluated the overall performance of the surrogate models in BO, the surrogate models were tested directly. This was done with a surrogate tester object that allows the user to define a test for any SMAC surrogate model. These tests exclusively evaluate predictions on a test set after training. Initialisation includes a list of surrogate models, a target function to test the surrogate models, the amount of training and validation data points, and a SMAC initial design generator. For an arbitrary iteration count, all included surrogate models are trained on the training data points and tested on the validation data points, which are both gathered from the target function using the initial design generator. The RMSE, GNLLL, maximum RMSE error, and correlation were collected when comparing the predicted values to the true values [Goe+07]. After collecting the values, the next iteration begins with the next seed. Following the last iteration, all metrics of the different iterations and surrogate models are output in result files.
For the optimisation test, facades were defined to be able to run a BO around the surrogate model. SMAC offers a facade structure in which all the necessary components for a run are defined. Included are the surrogate model, acquisition function, encoder, and many others. To run and test our new surrogate models and their many versions, these facades were implemented. Furthermore, for relevant tests, a consistent environment for different surrogate models was required. For example, before any HPO run begins, a collection of initial configuration tests are conducted. The configurations are randomly chosen, thus, this creates unnecessary random variables when comparing surrogate model performance. Instead, we chose to have a consistent group of initial configurations for all surrogate models during testing. A new SMAC facade that generates these initial configurations was required. It receives the intended final trial amount and runs the correct amount of initial configurations. The results were then stored in run history JSON files. These were then combined using custom scripts to accumulate to large run history files. This also allowed for the run history from any facade to be used as initial configuration data.
Once the facades containing the custom surrogate models receive these initial configurations, they directly begin with the BO. However, importing external initial configurations was not directly supported, so a workaround was implemented. In essence, SMAC allows for external run histories to be used. However, this is not easily achieved with different components of a facade that must be changed to accept them. To accomplish this, a subclass generator adds the necessary adjustments to an arbitrary facade. This avoided the alternative in which each facade would require a sub-class to be defined in which all components are correctly adjusted to accept the external run history. Equipped with the subclass generator, any facade could be forced to use an arbitrary run history for its initial configurations.
Using the customisable initial configurations, surrogate models were evaluated and compared with one another. The actual runs were executed on the High-Performance Computing (HPC) cluster CLAIX. To start the SMAC runs efficiently, jobs were started in batches using zsh shell scripts.

A config.json file contains the version names, facades, and any other important information describing what jobs are to be started. A shell script then reads the file and starts the jobs according to the settings. Our main.py accepts a variety of arguments, allowing for a nearly complete customised definition of the intended behaviour for a given job. Included amongst the arguments are the facade name, surrogate model version, a boolean to dictate the use of initial configurations, the dataset and model to be used, the number of workers, and the seed. This allows for the zsh script to easily designate arguments with which a job starts. Following execution, jobs output their results in JSON files, which were later transferred to be analysed and assessed.

Chapter 6

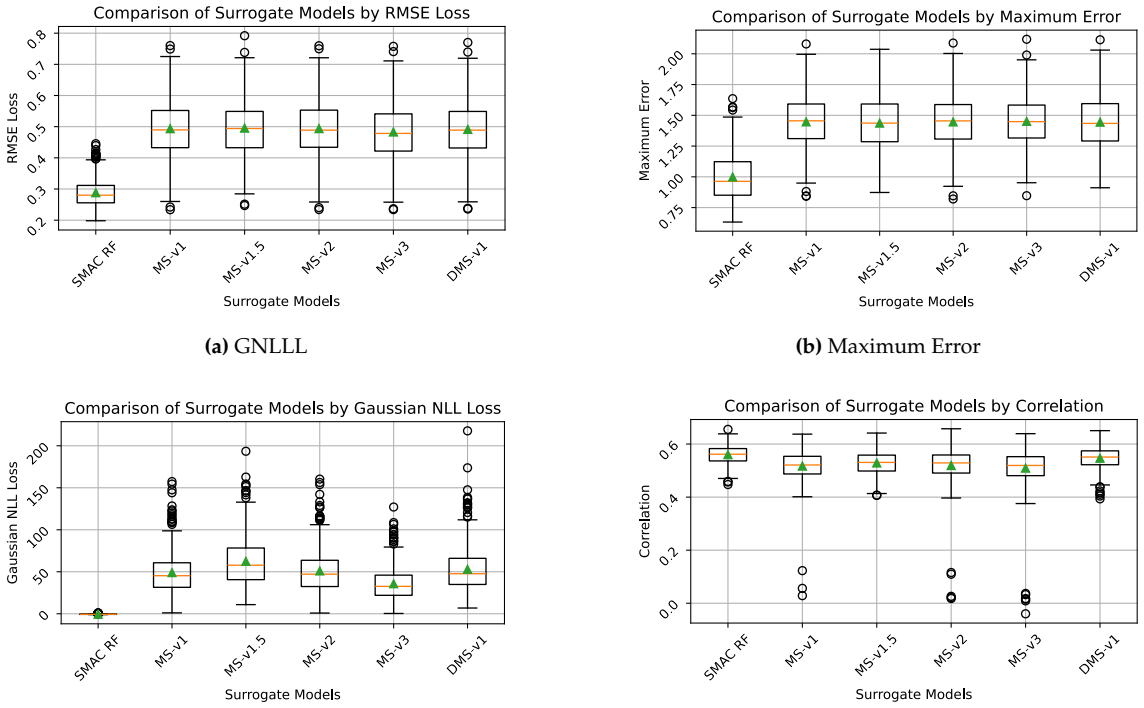
Experiments

Experimentation is the key to determining the performance and potential success of the surrogate models developed in this work. The computational cost must always be taken into account when conducting any experiments. To test the number of versions developed, it was not realistic to always test using the true use case. Training a large amount of DL-based forecasters was reserved for the end to compare the best-performing versions.
Testing began with the surrogate models alone. The aim of testing the surrogate models directly was to use the results to compare the accuracy of different versions before further testing. This was done using the Rosenbrock function with 20 hyperparameters so that a large amount of data was collectable. This helped minimise the influence of randomness when comparing the results between models. The testing also helped find any potential errors in the implementation and indicated the viability of the versions. Direct testing included training and testing the surrogate models on the Rosenbrock function. The performance of the resulting models was evaluated with RMSE, maximum RMSE error, correlation, and GNLLL. Work by Goel et al. [Goe+07] was used as a guide for the metric selection. The GNLLL was added as well to take predicted model variance into account. While loss functions and maximum error are common metrics for model accuracy, correlation can be useful when evaluating models specifically for BO. As already discussed, maximizers and acquisition functions are relied upon to pick the next configuration to run. If a high correlation with the true values exists, this might transfer over to the acquisition function values. Furthermore, it is important to recognize that, in essence, only the differences between acquisition function values are important when deciding on the next configuration. If the model predictions can produce acquisition function values that properly represent the relative differences between configurations, then this is a better signifier of final performance than a simple error function. The correlation might be an indicator for such a property, but the type of acquisition function used can easily distort any connection between acquisition value correlation and prediction correlation. This makes it extra difficult to compare functions, such as Expected Improvement, that do not have a real-world value. Therefore, these metrics can not definitively classify a single surrogate model as more performant than another during optimisation. However, they do offer a good approximate guideline for making comparisons between versions, especially since differences between them can be so small. Note that the conformal HPO method could not be included in the test as it does not work with a single prediction. The surrogate model includes many quantile regressors, of which only one predicts a mean. This mean predictor is not subject to any conformal methods, is only included for compatibility with SMAC, and isn't involved with the main maximisation process. Once tests included a sequential optimisation, the conformal surrogate model and its acquisition function could be tested.
In general, these optimisation test results were more expressive for all surrogate models as they tested performance in actual optimisation. We began with optimising the Rosenbrock function with 15 hyperparameters in which the value was minimised. Each instance received the same 750 initial configurations that were previously collected. Every surrogate model version tested, including the control for the combined model, was run multiple times with different seeds, and

each final incumbent was recorded. Comparison between surrogate models using this method used an eCDF plot to compare the distribution of best Rosenbrock values found.
Testing the optimisation on DL forecasters required actual data for training and evaluation. This data had to reflect the energy consumption data typically found in the real world and had to be large enough to train complex models. The ETDataset [Zho+21] was chosen due to its fulfilment of these conditions. It includes the oil temperature and 6 other features regarding the load on transformers in China. This reflects the data an energy supplier may already collect, which spans two years. The four sets included in the collection, ETTh1, ETTh2, ETTm1, and ETTm2, span the entire two years. ETTh1 and ETTh2 are two sets of hourly measurements, and ETTm1 and ETTm2 are sets containing measurements at 15-minute intervals.
Finally, two rounds of forecaster optimisations were done. The first included a larger set of initial configurations collected from some initial configurations and other optimisations. These sets ranged from 533 to 1000 configurations and included the RNN and LSTM models trained on the ETTh1 and ETTm1 data. These larger sets were chosen as some methods rely on splitting the training data. This splitting could have meant that the method only begins to outperform previous methods at higher initial configuration counts. It is also important to test with a low initial configuration, as this might have benefitted other methods and be representative of a specific use case. For very expensive target functions, one might not be able to afford many initial configurations, for example. These tests were only conducted using the RNN model on the ETTh1 and ETTm1 data. The number of initial configurations stayed consistent at 156 configurations. As some runs ended up optimising for more configurations due to differences in surrogate model training and target function execution, a maximum trial number was set during analysis. This meant that for a given context, such as training an RNN on the ETTh1 data with seed 0, only as many trials were considered as the run with the least amount of trials.

Chapter 7

Results



(a) GNLLL

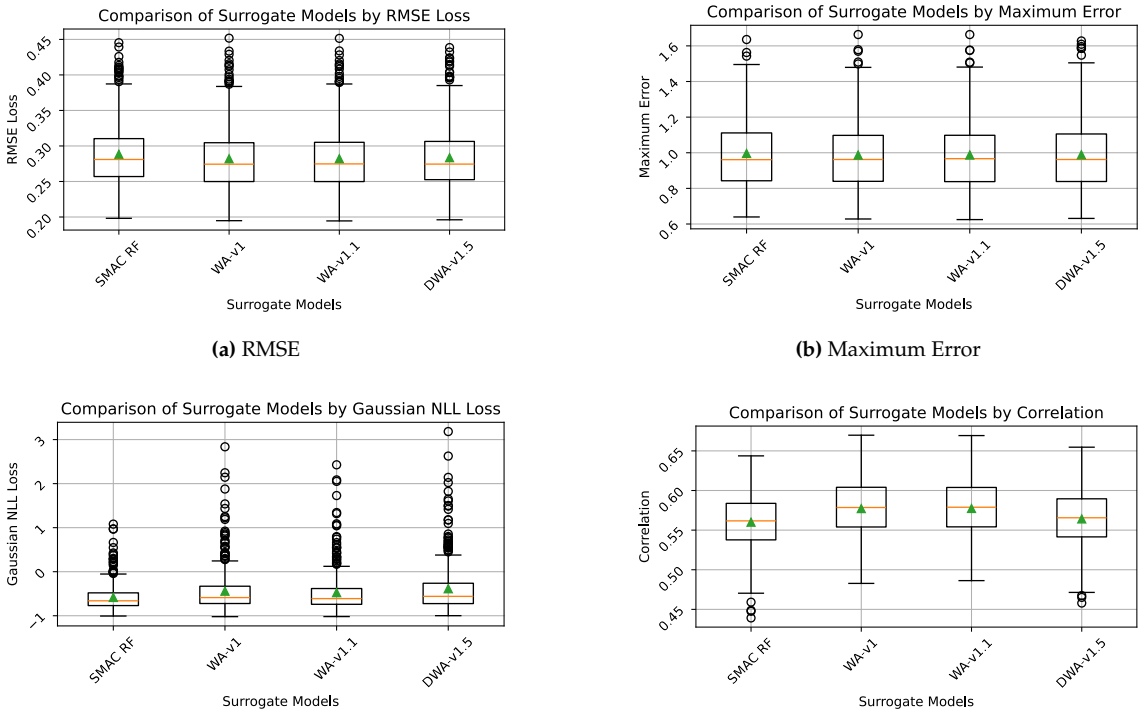
(b) Maximum Error

(c) GNLLL

(d) Correlation

Figure 7.1: Boxplots of 400 experiments measuring RMSE, Maximum Error, GNLLL, and Correlation for SMAC's RF, MS-v1, MS-v1.5, MS-v2, MS-v3, and DMS-v1 trained and validated on 600 points randomly sampled from the Rosenbrock Function

The results of the surrogate model accuracy tests are found in Figures 7.1 and 7.2. The results showed that RF performed best compared to the tested combined model versions. While some versions came close to matching the RF in some metrics, no version could consistently stay competitive. The test also revealed specifics about the setting choices of the combined model. For example, the simple broad model output transformation made little to no difference in the model performance. Furthermore, the difference in training order showed an influence in some metrics with a slightly higher GNLLL in the version that uses a broad model trained on fewer configurations to train the supplementary model. Interestingly, if the broad configuration space avoids the hyperparameters of the supplementary model, it has a slightly higher correlation but matches MS-v1 and MS-v2 in all other metrics. MS-v3 did not receive the variance from the output of the broad model and has a lower GNLLL than the others while matching them in the other metrics.



(a) RMSE

(b) Maximum Error

(c) GNLLL

(d) Correlation

Figure 7.2: Boxplots of 400 experiments measuring RMSE, Maximum Error, GNLLL, and Correlation for SMAC's RF, WA-v1, WA-v1.1, and DWA-v1.5 trained and validated on 600 points randomly sampled from the Rosenbrock Function

The model accuracy of the weighted aggregation was found to be closer to the RF, as expected. However, a very slight improvement in some metrics could be observed in contrast to the single-output combined models. While the RF performed better than the weighted aggregation models in GNLLL, no difference was found in the maximum RMSE, and the aggregation models proved to have a slight advantage in RMSE. The aggregation models also had the upper hand in correlation as long as their broad model received all hyperparameters. The predictions of the model with a mostly disjoint broad configuration space were less correlated to the true values compared to the predictions of the other aggregation-based models. Only a couple of differences could be observed among the different aggregation versions. The last difference amongst the tested models was the impact of using GNLLL for cross-validation, where WA-v1.1 had a slight advantage in the measured GNLLL over its peers.

Version	Average Incumbent Value
DMS-v4	425.99
Conformal	4453.32
DMS control	5711.49
DMS-v1.5.1	10487.37
DMS-v1.5.2	10509.22
WA-v1	13680.47
DMS-v1.5	18977.47
MS-v1	21205.46
SMAC RF	22408.09

Table 7.1: Incumbent values for different versions.

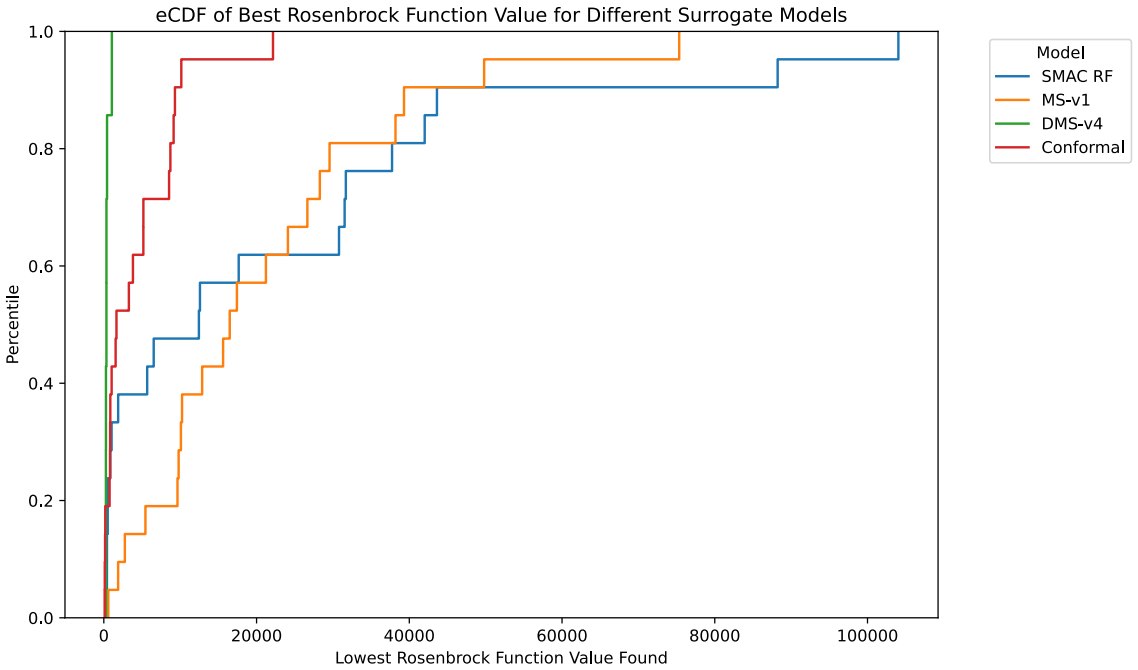
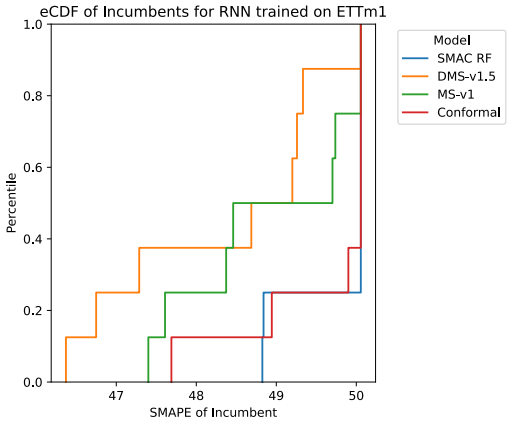


Figure 7.3: eCDF plot of the lowest Rosenbrock function values found during optimisation

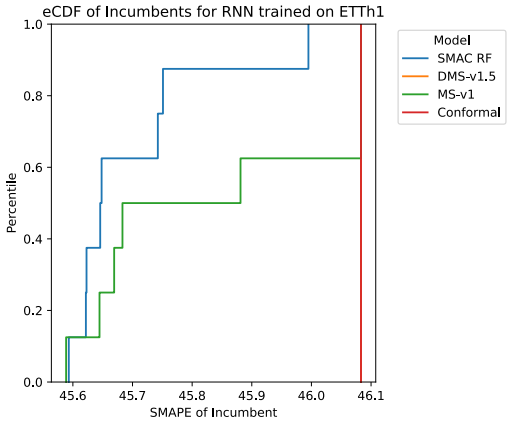
Based on the experience of the surrogate tests, a select number of models were chosen to participate in tests using the Rosenbrock function. In Figure 7.3, the eCDF plot can be found with the most important model versions[1]. These runs included the most variety in versions as this was relatively computationally inexpensive due to the Rosenbrock function and still indicative of how the models perform during an optimisation. This was also the first test the conformal method participated in, as it relies on a new acquisition function. One can see that using the default RF with SMAC led to various optimisation results. The RF outperformed many developed models in some percentiles. However, there were also percentiles in which it was the worst. MS-v1 showed a similarly high variety in resulting Rosenbrock values, with very few low values found compared to the others. Its average lowest Rosenbrock function value was very similar to that of the RF. The average Rosenbrock values found by each version are listed in Table 7.1. Versions that performed well were the conformal method and DMS-v4. These had much less variance between runs and an overall lower average value, with DMS-v4 sporting the lowest average.
Parallel to this, tests were done on actual forecaster optimisations, beginning with optimising an RNN using many initial configurations. When faced with the ETTh1 dataset, only MS-v1 and the standard RF with SMAC found new incumbents at similar rates and error scores. The conformal method and SMAC with DMS-v1.5 could not find any new incumbents. When viewing the model optimisation with the ETTm1 data, it is clear that MS-v1 performed better than any other method. While the RF could only lead to two unseen incumbents, the conformal method and the DMS-v1.5 with SMAC led to more success. The average SMAPE of the incumbents can be found in Table 7.4a[2].
To test versions on fewer initial configurations, the standard RNN model was trained on the datasets ETTh1 and ETTm1. The results can be seen in Figure 7.5. When models were forecasting ETTh1, the best results came when SMAC used the standard RF. As one can tell, DMS-v4 did fairly well compared to other new models but did not come close to matching the consistency of the RF. When the forecasting models were trained and tested on ETTh1, the RF did not keep its advantage. In these cases, it was the RF that could not find new incumbents. This is likely due to the incumbent of the initial configurations, which resulted in a low SMAPE. However, this difficulty

[1]See Figure A.2 for the complete eCDF plot with all versions and the control
[2]See Figure A.3 for the results of the optimisation of LSTM models on a high amount of configurations



Version	Average SMAPE
DMS-v1.5	48.36
MS-v1	48.92
Conformal	49.60
SMAC RF	49.75

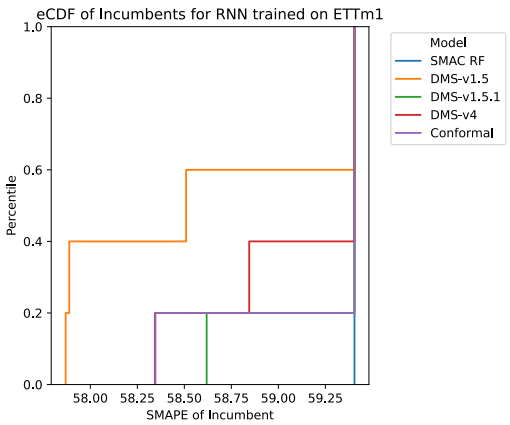
(a) RNN ETTm1 results with 936 initial configurations



Version	Average SMAPE
SMAC RF	45.70
MS-v1	45.84
DMS-v1.5	46.08
Conformal	46.08

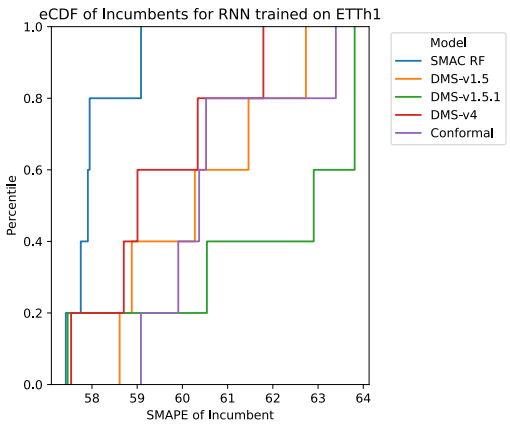
(b) RNN ETTh1 results 993 initial configurations

Figure 7.4: eCDF plots and average SMAPE values for RNN optimisations using different surrogate models



Version	Average SMAPE
DMS-v1.5	58.62
DMS-v4	59.08
Conformal	59.19
DMS-v1.5.1	59.25
SMAC RF	59.40

(a) RNN ETTm1 results with 156 initial configurations



Version	Average SMAPE
SMAC RF	58.02
DMS-v4	59.48
DMs-v1.5	60.39
Conformal	60.66
DMS-v1.5.1	61.71

(b) RNN ETTh1 results with 156 initial configurations

Figure 7.5: eCDF plots and average SMAPE values for RNN optimisations using different surrogate models

was overcome by DMS-v1.5, which found better configurations in most runs. DMS-v4 came in second once more, finding incumbents 2 out of 5 times compared to the other new surrogate models. Overall, the combined models seemed to perform well in optimising models for the ETTm1 data, while the RF performed well when the ETTh1 was used.

Hyperparameter	**Importance**
Optimizer Class	0.0785
Optimizer Learning Rate	0.0776
Training Output Length	0.0483
Learning Rate Scheduler Class	0.0246
Dropout	0.0108
Batch Size	0.0096
Hidden Dimension	0.0086
Input Chunk Length	0.0029
Loss Function	0.0013
Number of RNN Layers	0.0007
Validation Epoch Period	0.0003

Table 7.2: Average importance of each hyperparameter collected from 2000 configurations per model and data pair. The models considered were RNN and LSTM, trained on ETTh1 and ETTh2.

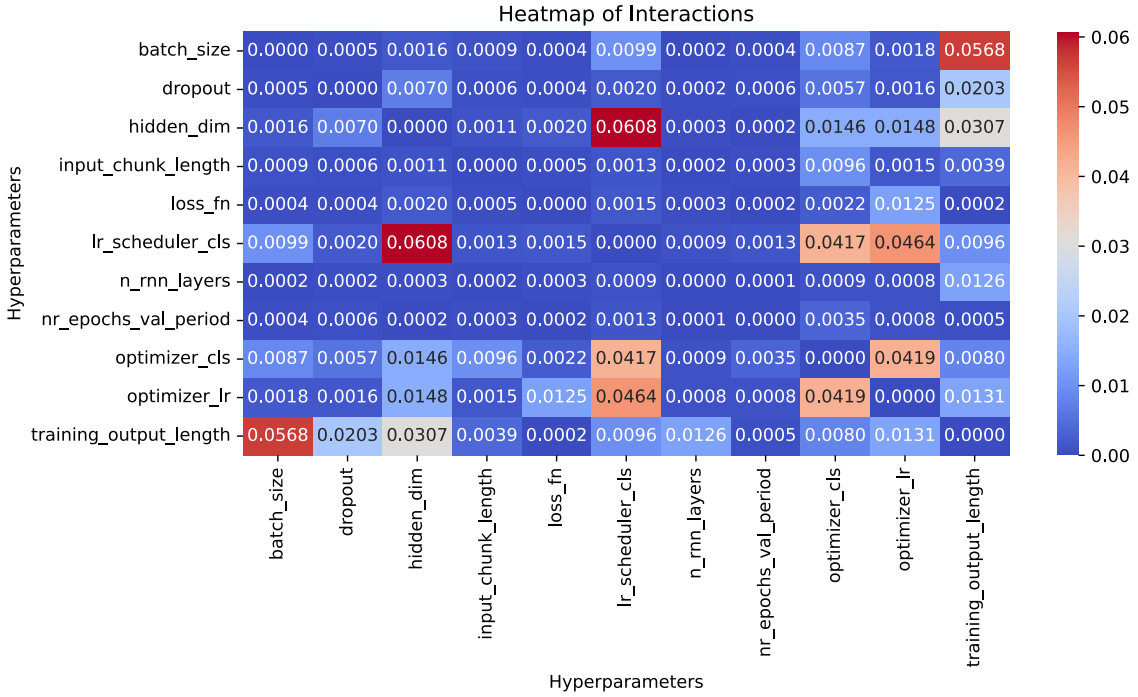
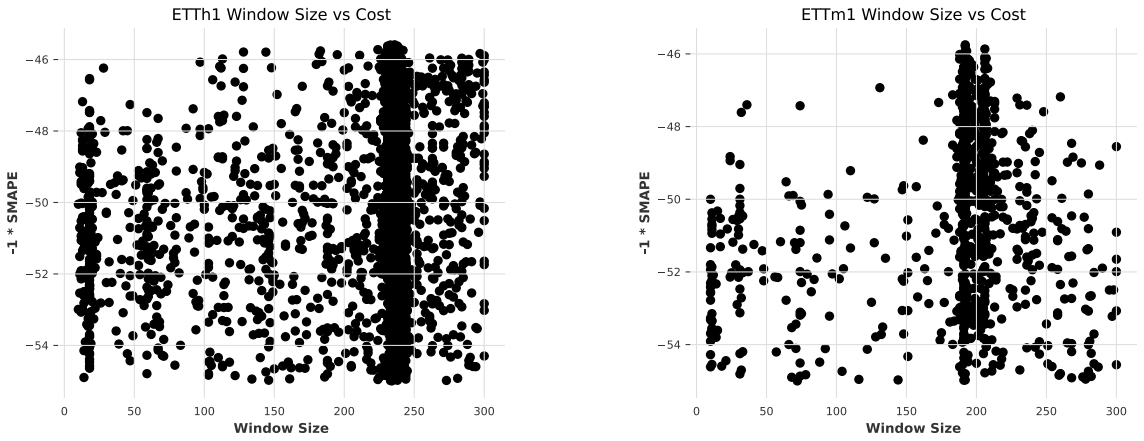


Figure 7.6: Heatmap of the average interactions between Hyperparameters of RNN and LSTM evaluated on ETTh1 and ETTm1

Finally, the configuration performance data helped draw conclusions about the window size hyperparameter, given all the optimisation tests. The average top 11 most important hyperparameters calculated by fANOVA can be seen in Table 7.2 The window size alone was not recognised by fANOVA as a very important hyperparameter placed in 8th when viewing the importance metrics of the unconditional hyperparameters. However, when viewing the complete configuration space of 35 hyperparameters, it had a bigger influence than the many conditionals, which were only assigned when the correct learning rate scheduler was chosen.

Looking at the interaction matrix of the LSTM trained on ETTm1 in Figure 7.6, one can learn about the interactions that took place. The interaction matrix showed that the window size did not seem to have any major interactions with many hyperparameters compared to other interactions. When viewing the interactions that it did have, it is easy to see that the optimiser class and the output length during training had the strongest relationship with the window size. This meant that it was most important to examine the window size with these hyperparameters when searching for specific influences on performance.
Using the many configurations gathered, the direct effect that the window size had on performance can be analysed. In Figure 7.7, one can recognise that the window size did not seem to have a monotonic effect on the performance metric. This means one could not simply pick a large or small window to maximise results. Recognising that the results were gathered from optimisation runs is also important. This may have caused sampling biases when a given window size range was considered promising. This would cause that range to be sampled more, increasing the chance of finding well-performing configurations with that window size. Viewing the density of the specific regions indicates the sampling frequency of the optimisation runs. While many data points were spread relatively evenly across the window size scale, many configurations have been sampled with a window size between 175 and 250. This indicates that surrogate models consistently recognised the potential for new incumbents in this region. In general, the complex relationship between window size and performance can also imply that the combination with other hyperparameters has a significant effect on the performance of a given configuration.



(a) ETTh1 **(b)** ETTm1

Figure 7.7: Plots of the window size to cost relationships of accumulated from models trained on ETTh1 and ETTm1

In Figure 7.8, the relationships between the window size and other hyperparameters can be examined. The cumulative data indicated that the window size had the strongest relationships with the hidden dimension count, the optimiser learning rate, and the training output length. This coincides with the findings of the interaction heatmap in Figure 7.6. One can find low-cost clustering in the scatter plots with the window size. This indicates that a well-performing model would result only once both hyperparameters were set in accordance with one another. For example, while only comparing performance with window size, one might conclude that a low window size is just as viable as a larger size for training models on the ETTh1 data. However, when viewing the interactions with other hyperparameters, it is clear that the window size works best with other hyperparameters when it is close to its upper limit.

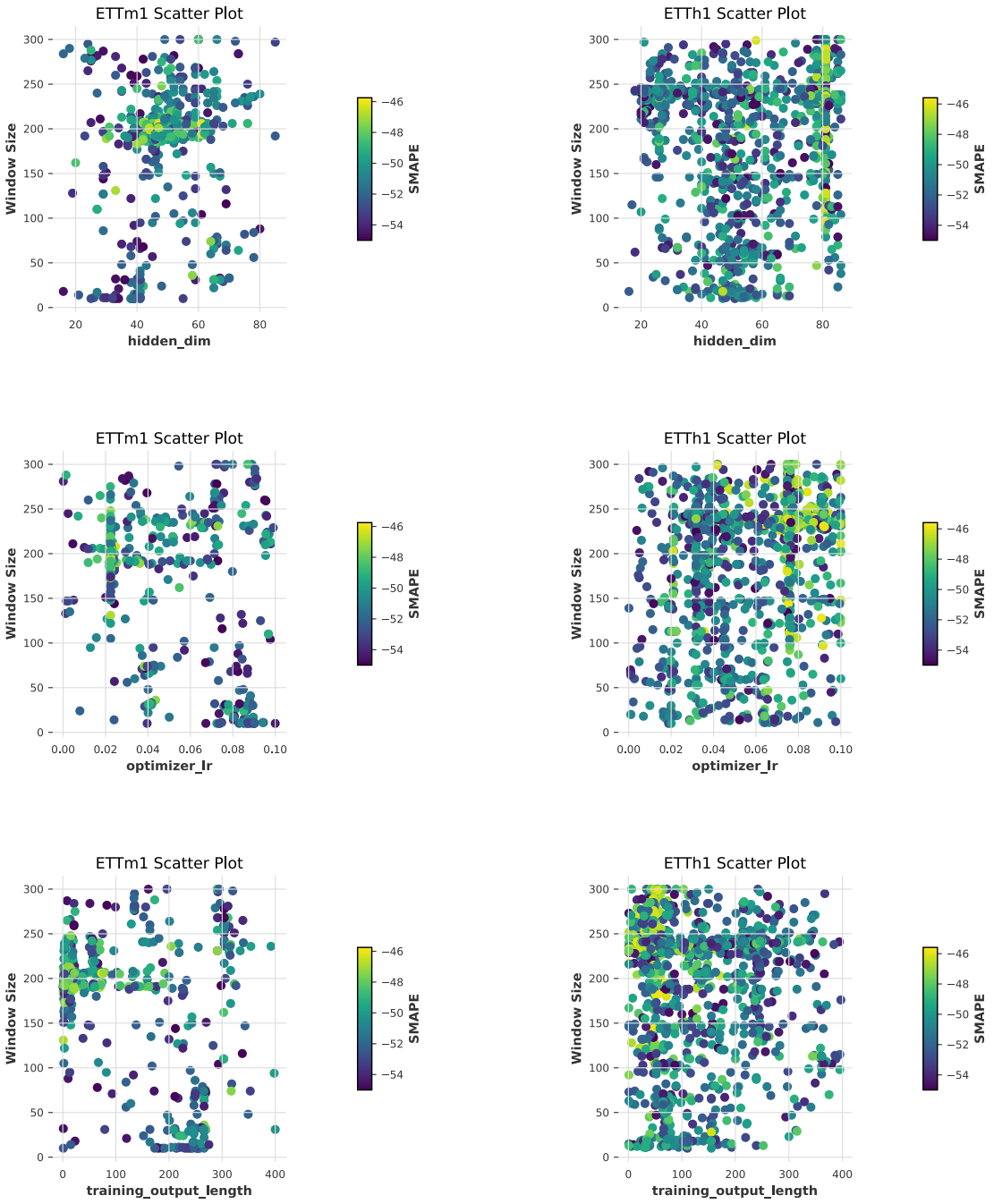


Figure 7.8: Scatter plot of hyperparameters with the most clear interactions with the window size

Chapter 8

Conclusion

In this work, we looked at the window size as an influential hyperparameter to see how it affects DL-based forecasters. The popular fANOVA library was used for hyperparameter analysis in which we observed how the importance and interactions are distributed amongst the configuration space of forecasters. It was deduced that the influence of the window size alone is not the strongest compared to other hyperparameters, but it is still important enough to be carefully considered when implementing a forecaster. Its interactions with other select hyperparameters illuminate the complex nature of the hyperparameter and the need for non-trivial selection methods. Using our findings, we concluded that potential exists to focus the optimisation process on the window size and important, unconditional, numerical hyperparameters using a GP as a surrogate model. These hyperparameters proved to have relatively few interactions, which indicated that they did not depend heavily on each other's values. It was concluded that a good selection could be made if the window size was only separately optimised from hyperparameters with which it had minimal interactions. To cover other, less important hyperparameters, an RF was used to supply the GP with an assessment of the rest of the configuration. This combined surrogate model idea spawned many variations, which helped tweak small settings to result in a well-performing optimisation process. The integration took place in the already popular BO implementation SMAC, which also helped make systematic comparisons between new surrogate models and the existing sole RF. Furthermore, a new conformal surrogate model based on the ideas of Salinas et al. in [Sal+23] was integrated into SMAC for testing. While the conformal optimisation methods explored were not sufficient enough to display any major improvements over existing methods, it was found that the combined surrogate model was able to match or beat RF in some cases. This showed that the concept of a separated hyperparameter space for the surrogate models in optimisers has promise to assist in finding better hyperparameters. Regarding the analysis of hyperparameters, it was found that some weak interactions do exist between the window size and other hyperparameters. Thus, no clear window size range containing the most promising sizes could be concluded. Additionally, due to the high variance in performance amongst all window sizes observed, it is essential to consider the other hyperparameters as well.

Chapter 9

Limitations and Future Work

Open questions and ideas remain to extend and improve the results presented in this work. For example, when setting the configuration space, a window size range of (0,300) was assumed to suffice. In hindsight, a much larger range would allow conclusions to be drawn about the existence of an upper bound at which model performance falls. Furthermore, due to uneven sampling caused by the optimisation methods used to collect data, some uncertainty exists surrounding the conclusions gained from the scatter plots. A density-based normalisation to remove any sampling biases would be helpful. More research into the efficacy of the Rosenbrock function to evaluate HPO methods should be conducted, as a large discrepancy could be found between the tested performance on the function and the actual performance on forecaster optimisation.
When viewing the developed HPO methods, an expansion of version testing is, at first glance, the most reasonable due to differences found between performance on the Rosenbrock function and actual forecasters. This would allow for more educated guesses as to which settings might profit from being combined with newer versions. Regarding the combined model specifically, one could explore more approaches to integrate the combined method into BO. As mentioned in [Sal+23], GPs outperform other surrogate models during optimisation with very few observations. One could postulate that using a combined method instead of a random model to generate initial configurations would give the optimisation process a head start. Once enough data has been collected, a standard RF can be used as a surrogate model as normal. Lastly, calculating the configuration space partition for the supplementary model could be done with an optimiser instead. Finding the best partition was solved using brute force; however, with a higher number of numerical hyperparameters considered, using a solver may be beneficial.
Regarding the conformal methods implemented in SMAC, one must recognise a few areas of improvement. A different acquisition function could be implemented rather than using a rudimentary method like Thompson Sampling. Using the quantile regression, one could adapt Expected Improvement to accept an approximated distribution function. One could also consider using the existing RF implementation and combine it with conformal methods to adjust the variance of the model using a calibration set. Finally, the conformal surrogate model could implement TCP instead of ICP. Currently, the surrogate model uses data splitting to produce a training and calibration set. This is computationally very efficient; however, it reduced the size of the training set. Alternatively, TCP ensures the same functionality with the complete data being used for training. This comes at a cost of computation, which becomes more expensive the more data samples you have. This would offer a conformal alternative for cases in which data is very sparse due to the expensive nature of the target function.

Acknowledgements

The computing resources were granted by RWTH Aachen University under project rwth1701. The authors gratefully acknowledge the computing time provided to them at the NHR Center NHR4CES at RWTH Aachen University (project number: p0023880). This is funded by the Federal Ministry of Education and Research, and the state governments participating on the basis of the resolutions of the GWK for national high performance computing at universities (www.nhr-verein.de/unsere-partner).

References

[CFS04] Michael P Clements, Philip Hans Franses, and Norman R Swanson. "Forecasting Economic and Financial Time-Series with Non-Linear Models". In: *International Journal of Forecasting* 20.2 (2004), pp. 169–183.

[AD21] Mohammadreza Akbari and Thu Nguyen Anh Do. "A Systematic Review of Machine Learning in Logistics and Supply Chain Management: Current Trends and Future Directions". In: *Benchmarking: An International Journal* 28.10 (2021), pp. 2977–3005.

[Pri+25] Ilan Price et al. "Probabilistic weather forecasting with machine learning". In: *Nature* 637.8044 (2025), pp. 84–90.

[FM20] Carlo Fezzi and Luca Mosetti. "Size Matters: Estimation Sample Length and Electricity Price Forecasting Accuracy". In: *The Energy Journal* 41.4 (2020).

[SMR21] Nivethitha Somu, Gauthama Raman MR, and Krithi Ramamritham. "A Deep Learning Framework for Building Energy Consumption Forecast". In: *Renewable and Sustainable Energy Reviews* 137 (2021), p. 110591.

[Zai+21] Ameema Zainab et al. "Big Data Management in Smart Grids: Technologies and Challenges". In: *IEEE Access* 9 (2021), pp. 73046–73059.

[STN18] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. "A Comparison of ARIMA and LSTM in Forecasting Time Series". In: *Proceedings of the 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2018, pp. 1394–1401.

[SSR19] Shivani, K.S. Sandhu, and Anil Ramachandran Nair. "A Comparative Study of ARIMA and RNN for Short Term Wind Speed Forecasting". In: *Proceedings of the 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2019, pp. 1–7.

[Dia+17] Gonzalo I Diaz et al. "An Effective Algorithm for Hyperparameter Optimization of Neural Networks". In: *IBM Journal of Research and Development* 61.4/5 (2017), pp. 9–1.

[IJR17] Atsushi Inoue, Lu Jin, and Barbara Rossi. "Rolling Window Selection for Out-of-Sample Forecasting with Time-Varying Parameters". In: *Journal of Econometrics* 196.1 (2017), pp. 55–67.

[SJN22] Jimeng Shi, Mahek Jain, and Giri Narasimhan. *Time Series Forecasting (TSF) Using Various Deep Learning Models*. 2022. arXiv: `2204.11115 [cs.LG]`.

[Den+22] Difan Deng et al. "Efficient Automated Deep Learning for Time Series Forecasting". In: *Proceedings of Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 2022, pp. 664–680.

[Bis+23] Bernd Bischl et al. “Hyperparameter Optimization: Foundations, Algorithms, Best Practices, and Open Challenges”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 13.2 (2023), e1484.

[AB22] Anastasios N. Angelopoulos and Stephen Bates. *A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification*. 2022. arXiv: 2107.07511 [cs.LG].

[Sal+23] David Salinas et al. “Optimizing Hyperparameters with Conformal Quantile Regression”. In: *Proceedings of International Conference on Machine Learning*. 2023.

[LF87] A Lapedes and R Farber. “Nonlinear signal processing using neural networks: Prediction and system modelling”. In: *Proceedings of the 1st IEEE International Conference on Neural Networks*. June 1987.

[ZPH98] Guoqiang Zhang, B. Eddy Patuwo, and Michael Y. Hu. “Forecasting with Artificial Neural Networks: The State of the Art”. In: *International Journal of Forecasting* 14.1 (1998), pp. 35–62.

[Liu+17] Weibo Liu et al. “A Survey of Deep Neural Network Architectures and Their Applications”. In: *Neurocomputing* 234 (2017), pp. 11–26.

[Vas+17] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762.

[Lei+20] Charles E. Leiserson et al. “There’s Plenty of Room at the Top: What Will Drive Computer Performance After Moore’s Law?” In: *Science* 368.6495 (2020), eaam9744.

[Zho+21] Haoyi Zhou et al. “Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting”. In: *Proceedings of the 35th AAAI Conference on Artificial Intelligence, Virtual Conference*. Vol. 35. 12. 2021, pp. 11106–11115.

[Her+22] Julien Herzen et al. “Darts: User-Friendly Modern Machine Learning for Time Series”. In: *Journal of Machine Learning Research* 23.124 (2022), pp. 1–6.

[Ore+20] Boris N. Oreshkin et al. *N-BEATS: Neural Basis Expansion Analysis for Interpretable Time Series Forecasting*. 2020. arXiv: 1905.10437 [cs.LG].

[Lim+21] Bryan Lim et al. “Temporal Fusion Transformers for Interpretable Multi-Horizon Time Series Forecasting”. In: *International Journal of Forecasting* 37.4 (2021), pp. 1748–1764. ISSN: 0169-2070.

[Als+22] Ahmad Alsharef et al. “Review of Machine Learning and AutoML Solutions to Forecast Time-Series Data”. In: *Archives of Computational Methods in Engineering* 29.7 (2022), pp. 5297–5311.

[AZY20] Tanveer Ahmad, Hongcai Zhang, and Biao Yan. “A Review on Renewable Energy and Electricity Requirement Forecasting Models for Smart Grid and Buildings”. In: *Sustainable Cities and Society* 55 (2020), p. 102052.

[QAR24] Momina Qureshi, Masood Ahmad Arbab, and Sadaqat ur Rehman. “Deep Learning-Based Forecasting of Electricity Consumption”. In: *Scientific Reports* 14.1 (2024), p. 6489.

[Kau+22] Devinder Kaur et al. “Energy Forecasting in Smart Grid Systems: Recent Advancements in Probabilistic Deep Learning”. In: *IET Generation, Transmission & Distribution* 16.22 (2022), pp. 4461–4479.

[Tri15] Artur Trindade. *ElectricityLoadDiagrams20112014*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C58C86. 2015.

[HHL14] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. "An Efficient Approach for Assessing Hyperparameter Importance". In: *Proceedings of International Conference on Machine Learning*. 2014, pp. 754–762.

[Hoo07] Giles Hooker. "Generalized Functional ANOVA Diagnostics for High-Dimensional Functions of Dependent Variables". In: *Journal of Computational and Graphical Statistics* 16.3 (2007), pp. 709–732.

[ESL23] Arik Ermshaus, Patrick Schäfer, and Ulf Leser. "Window Size Selection in Unsupervised Time Series Analytics: A Review and Benchmark". In: *Proceedings of International Workshop on Advanced Analytics and Learning on Temporal Data*. 2023, pp. 83–101.

[HHL11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Proceedings of Learning and Intelligent Optimization*. 2011, pp. 507–523. ISBN: 978-3-642-25566-3.

[Lin+22] Marius Lindauer et al. "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization". In: *Journal of Machine Learning Research* 23.54 (2022), pp. 1–9.

[FH19] Matthias Feurer and Frank Hutter. In: *Automated Machine Learning: Methods, Systems, Challenges*. Springer International Publishing, 2019, pp. 3–33. ISBN: 978-3-030-05318-5.

[GH20] Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. "Dealing with Categorical and Integer-Valued Variables in Bayesian Optimization with Gaussian Processes". In: *Neurocomputing* 380 (2020), pp. 20–35.

[Egg+13] Katharina Eggensperger et al. "Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters". In: *Proceedings of NIPS Workshop on Bayesian Optimization in Theory and Practice*. Vol. 10. 3. 2013, pp. 1–5.

[Ber+11] James Bergstra et al. "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems* 24 (2011).

[RG20] Thomas N. Rincy and Roopam Gupta. "Ensemble Learning Techniques and Its Efficiency in Machine Learning: A Survey". In: *Proceedings of the 2nd International Conference on Data, Engineering and Applications (IDEA)*. 2020, pp. 1–6.

[Goe+07] Tushar Goel et al. "Ensemble of Surrogates". In: *Structural and Multidisciplinary Optimization* 33 (2007), pp. 199–216.

[Wol92] David H. Wolpert. "Stacked Generalization". In: *Neural Networks* 5.2 (1992), pp. 241–259.

[CL11] Olivier Chapelle and Lihong Li. "An Empirical Evaluation of Thompson Sampling". In: *Proceedings of Advances in Neural Information Processing Systems*. Vol. 24. 2011.

[Tho33] William R. Thompson. "On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples". In: *Biometrika* 25.3-4 (1933), pp. 285–294.

[Kan+18] Kirthevasan Kandasamy et al. "Parallelised Bayesian Optimisation via Thompson Sampling". In: *Proceedings of International Conference on Artificial Intelligence and Statistics*. 2018, pp. 133–142.

[Elh+19] Ouassim Elhara et al. *COCO: The Large Scale Black-Box Optimization Benchmarking (bbob-largescale) Test Suite*. 2019. arXiv: 1903.06396 [math.OC].

[Che+22] Yutian Chen et al. "Towards Learning Universal Hyperparameter Optimizers with Transformers". In: *Proceedings of Advances in Neural Information Processing Systems*. Vol. 35. 2022, pp. 32053–32068.

[Tan+21] Laurits Tani et al. "Evolutionary Algorithms for Hyperparameter Optimization in Machine Learning for Application in High Energy Physics". In: *The European Physical Journal C* 81.2 (Feb. 2021). ISSN: 1434-6052.

[SQ06] Yun-Wei Shang and Yu-Huang Qiu. "A Note on the Extended Rosenbrock Function". In: *Evolutionary Computation* 14.1 (Mar. 2006), pp. 119–126. ISSN: 1063-6560.

[HHL10] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration (Extended Version)". In: *Technical Report TR-2010–10, University of British Columbia, Computer Science, Tech. Rep.* (2010).

[JSW98] Donald R. Jones, Matthias Schonlau, and William J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13 (1998), pp. 455–492.

[JKB94] Norman Lloyd Johnson, Samuel Kotz, and N. Balakrishnan. *Continuous Univariate Distributions. I / Norman L. Johnson ; Samuel Kotz ; N. Balakrishnan*. 2nd ed. New York [u.a]: Wiley, 1994. ISBN: 0471584959.

[Lin+19] Marius Lindauer et al. "BOAH: A Tool Suite for Multi-Fidelity Bayesian Optimization & Analysis of Hyperparameters". In: (2019). arXiv: 1908.06756 [cs.LG].

[HA18] Rob J Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2018.

Appendix A

Additional material

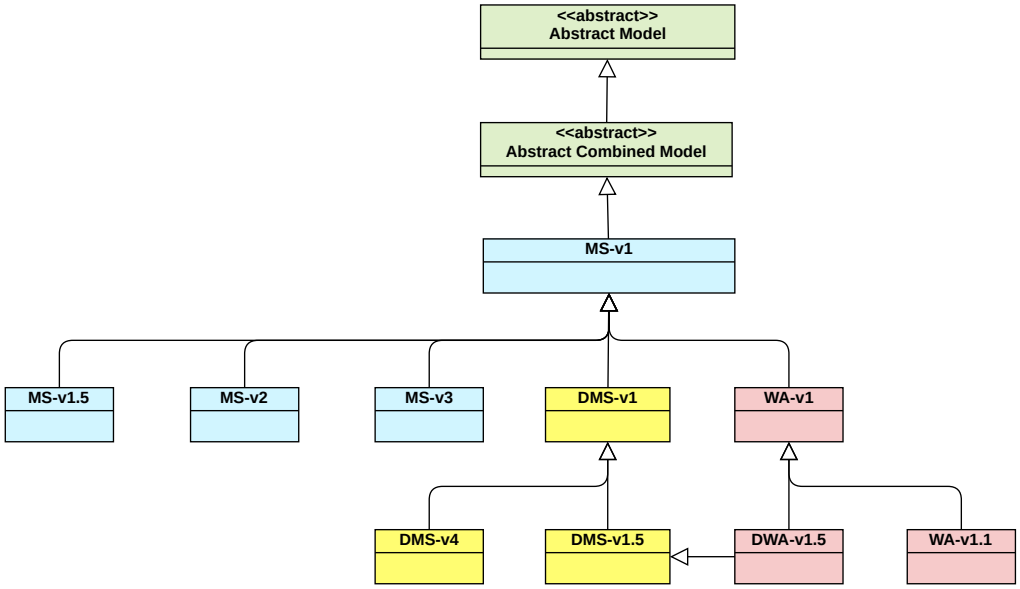


Figure A.1: Visualisation of the class hierarchy of different model versions

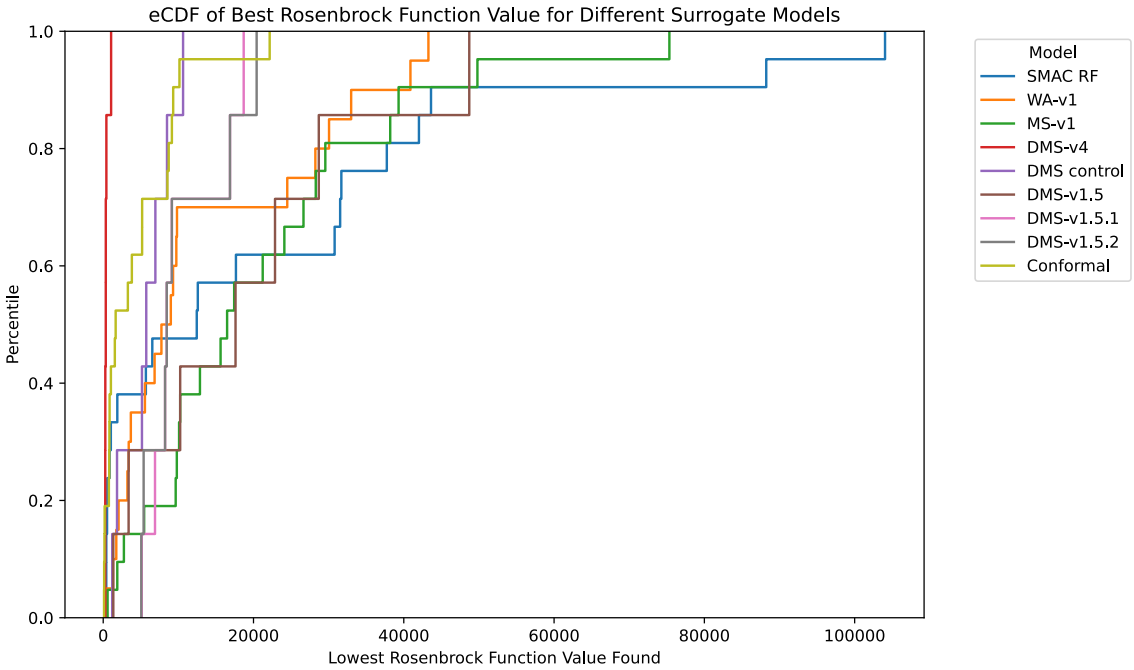
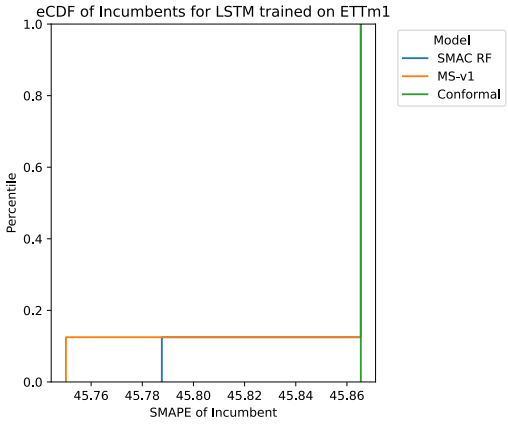
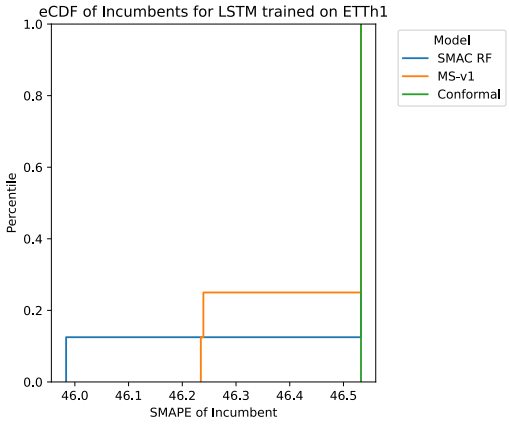


Figure A.2: eCDF plot of the lowest Rosenbrock function values found during optimisation



Version	Average SMAPE
MS-v1	45.85
SMAC RF	45.86
Conformal	45.86

(a) LSTM ETTm1 results with 540 initial configurations



Version	Average SMAPE
MS-v1	46.46
SMAC RF	46.46
Conformal	46.53

(b) LSTM ETTh1 results with 976 initial configurations

Figure A.3: eCDF plots and average SMAPE values for LSTM optimisations using different surrogate models

Version	Encoding	Loss Function	Broad Output Transform	Training Data Split
		Broad Space	**Aggregate Output**	**Training Order**
MS-v1	log Encoding	N/A	No Change	No Split
		Complete Coverage	No Aggregation	Supplementary first
MS-v1.5	log Encoding	N/A	No Change	No Split
		Complete Coverage	No Aggregation	Broad first
MS-v2	log Encoding	N/A	Range Transform	No Split
		Complete Coverage	No Aggregation	Broad first
MS-v3	log Encoding	N/A	No Variance Transform	No Split
		Complete Coverage	No Aggregation	Broad first
DMS-v1	log Encoding	N/A	No Change	No Split
		Mostly Disjoint	No Aggregation	Supplementary first
DMS-v1.5	log Encoding	N/A	No Change	No Split
		Mostly Disjoint	No Aggregation	Broad first
DMS-v1.5.1	Mixed Encoding	N/A	log-Normal Transform 1	No Split
		Mostly Disjoint	No Aggregation	Broad first
DMS-v1.5.2	Mixed Encoding	N/A	log-Normal Transform 2	No Split
		Mostly Disjoint	No Aggregation	Broad first
DMS-v4	log Encoding	N/A	No Change	50/50 Split
		Mostly Disjoint	No Aggregation	Broad first
WA-v1	log Encoding	RMSE	No Change	No Split
		Complete Coverage	Weighted Aggregation	Supplementary first
WA-v1.1	log Encoding	GausianNLL Loss	No Change	No Split
		Complete Coverage	Weighted Aggregation	Supplementary first
DWA-v.1.5	log Encoding	RMSE	No Change	No Split
		Mostly Disjoint	Weighted Aggregation	Broad first

Table A.1: Table of the different combined model versions and the distinguishing settings of each.

Hyperparameter	Type	Range/Choices	Default Value
General Parameters			
input_chunk_length	Uniform Integer	[10, 300]	20
model_type	Categorical	{RNN, LSTM, GRU}	LSTM
hidden_dim	Uniform Integer	[16, 86]	32
n_rnn_layers	Uniform Integer	[2, 5]	2
dropout	Uniform Float	[0.0, 0.5]	0.2
batch_size	Uniform Integer	[16, 128]	32
loss_fn	Categorical	{mse, l1loss, huber, smoothl1loss}	mse
training_output_length	Uniform Integer	[1, 400]	80
optimizer_cls	Categorical	{Adagrad, Adam, AdamW, LBFGS, RMSprop, SGD}	Adam
optimizer_lr	Uniform Float	[1e-5, 1e-1]	1e-3
lr_scheduler_cls	Categorical	{None, CosineAnnealingWarmRestarts ExponentialLR, PolynomialLR, ReduceLROnPlateau, StepLR, CyclicLR, OneCycleLR, LinearLR CosineAnnealingLR, ConstantLR}	None
nr_epochs_val_period	Uniform Integer	[1, 10]	1
StepLR Parameters			
step_size	Uniform Integer	[1, 10]	5
gamma	Uniform Float	[0.1, 0.9]	0.5
ConstantLR Parameters			
factor	Uniform Float	[0.1, 0.9]	0.5
total_iters	Uniform Integer	[1, 10]	5
LinearLR Parameters			
start_factor	Uniform Float	[0.1, 1.0]	1.0
end_factor	Uniform Float	[0.01, 1.0]	0.1
total_iters_linear	Uniform Integer	[1, 10]	5
ExponentialLR Parameters			
gamma_exp	Uniform Float	[0.8, 0.99]	0.9
PolynomialLR Parameters			
power	Uniform Float	[0.5, 2.0]	1.0
total_iters_poly	Uniform Integer	[1, 10]	5
CosineAnnealingLR Parameters			
t_max	Uniform Integer	[1, 10]	5
eta_min	Uniform Float	[0.0, 0.1]	0.0
CosineAnnealingWarmRestarts Parameters			
t_0	Uniform Integer	[1, 10]	5
t_mult	Uniform Integer	[1, 2]	1
eta_min_warm	Uniform Float	[0.0, 0.1]	0.0
ReduceLROnPlateau Parameters			
factor_rop	Uniform Float	[0.1, 0.9]	0.5
patience_rop	Uniform Integer	[1, 10]	3
mode_rop	Categorical	{min, max}	min
CyclicLR Parameters			
base_lr_cyc	Uniform Float	[1e-5, 1e-3]	1e-4
max_lr	Uniform Float	[1e-3, 1e-1]	1e-2
gamma_cyc	Uniform Float	[0.1, 0.9]	0.5
OneCycleLR Parameters			
total_steps_oc	Uniform Integer	[100, 1000]	500
pct_start_oc	Uniform Float	[0.1, 0.5]	0.3

Table A.2: Table of all hyperparameters and their according bounds and values