Universiteit Utrecht

Department of Information and Computing Science
Utrecht University

Artificial Intelligence master thesis

Combining static and temporal features with attention-mechanisms for classification and forecasting

First examiner:
Xixi Lu

Candidate:
Hendrik Carel Krijt

Second examiner:
Thijs van Ommen

Daily supervisor:
Wouter van der Waal

April 30, 2024

Abstract

Utilizing both static and time series data can enhance the performance of machine learning models. However, existing methods of concatenating data lead to high dimensionality and learning of noise. In this thesis, we investigate the addition of an attention mechanism to learn the correlation between static and time series data to mitigate this problem and increase model accuracy. In a multiple case study consisting of 2 real-life medical cases, 1 public dataset, and 1 synthetic dataset, we find similar or better results when using an architecture with an attention mechanism, compared to similar hybrid or meta architectures without it, particularly in sequence forecasting tasks. Additionally, we inspect whether the attention weights align with key events and can reveal structural dependencies within the data. While the attention weights reflected the structural dependencies in our synthetic Fibonacci sequence forecasting experiment, they did not align with key events in our real-life cystic fibrosis improvement classification experiment. We conclude that adding an attention mechanism can improve or maintain performance in forecasting problems. We provide suggestions for additional research into evaluating the explainability of attention mechanisms.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations:

- My daily supervisor, Wouter van der Waal, for his expert guidance over the last year. Thank you for your patience, advice, motivation and always having an answer.
- The Wilhelmina Kinderziekenhuis for allowing us access to the cystic fibrosis data and study facilities.
- Dr. Marlou Bierlaagh and Dr. Danya Muilwijk for answering all our data-related questions and going the extra mile in organizing access to the data. Thank you for the good work you are doing in the hospital.
- My thesis buddy Sander for brainstorming over countless ideas and architectures and making the dull moments interesting.
- My girlfriend, flatmate, friends and family for their support and much needed understanding. Thank you for listening to my endless ramblings about attention mechanisms.
- To God, for all the incredible opportunities I have been given in my life.

Contents

1. Introduction

In today's data-driven age, machine learning models can provide a great advantage in decision-making across various domains. Machine learning is a field in computer science in which algorithms and techniques are explored to create models which automate solutions for complex problems that are challenging to address using conventional programming methods [1]. For instance, a machine learning model can be trained to detect early signs of diabetic retinopathy by analyzing retinal images. The model can learn to identify patterns and abnormalities that might indicate the presence of the disease, assisting doctors in making more accurate diagnoses [2]. Predictive machine learning is a subset of machine learning which focuses on making reliable predictions about future events or outcomes. An example is forecasting future demand for a particular product. Machine learning models can analyze historical sales data and market trends to forecast the expected demand for the product [1]. This information can be used to optimize production and inventory management ensuring companies can meet customer demands.

Recent advancements in machine learning have been significantly influenced by deep learning, a subset of machine learning that employs neural networks with multiple layers to analyze various forms of data [3]. However, it requires a substantial amount of data and functions as "black boxes" – a model which makes predictions without an interpretable mechanism for explaining the process behind those predictions [4], making it less transparent [5].

That being said, as data availability remains a determining factor in the success of any machine learning model [1, 6], a lack of data poses a challenge to the utilization of machine learning models. The challenge originates from the idea that having a larger dataset captures a more accurate representation of the problem domain which, in turn, stems from the law of large numbers, a theorem in probability and statistics which states that the average result

from repeating an experiment multiple times will better approximate the true or expected underlying result [7]. Hence, if there is not enough data, the model is likely to overfit, that is to describe features that arise from noise or variance in the data, rather than the underlying distribution from which the data was drawn [8], leading to loss of accuracy on unseen data [9].

Solutions to this problem do exist, such as early stopping [10], penalty methods [11], using simpler models, and of course, including more data. Including more of the same data is often not possible, but including diverse data sources and data types such as temporal data, sensor readings, or images, have the potential to improve predictive performance.

Most machine learning models are engineered to work with one specific type of data, namely static data for static models, and time series models for temporal data, but real-world situations often include both [12]. The co-occurrence of these features range from electronic health records [13] and financial markets to environmental monitoring and traffic analyses [14, 15]. Static features include features which do not change over a given period such as a person's date of birth, blood type and genetics whilst temporal features include features which track the change of processes through time-stamped observations, also referred to as time steps, such as the daily price of a stock, the outside temperature and a diabetic's sugar levels [12].

Typically, data integration methods for predictive models involve combining static and temporal data by first transforming the temporal data into statistical features. For instance, in the study by Huddar et al. [16], they extracted statistical measures such as the mean, standard deviation, range, and skewness from temporal numerical measurements. In another approach described by Wang et al. [17], temporal features were converted into static features that describe the frequency components of the data using the Fourier transform. These transformations reduce the amount of data to be processed and effectively eliminate the time component, allowing them to be treated as additional static variables. However, by design, any simplification of data leads to a loss of information, as the model fails to completely grasp the inherent relationship between time and the temporal data, which is especially

important when considering time-specific predictions.

Similarly, static data can be added to time series models by concatenating the static features to the temporal data. A popular deep learning approach to time series models are Recurrent Neural Networks (RNNs) [18], which specialize in processing sequential data. Common ways of adding static data to RNN models are to either feed the static data to the initial hidden state or simply concatenate the static data at every timestep [19]. Whilst this does preserve the time component, it does have added complexity and could make the model more susceptible to learning noise [20].

A popular method to increase performance as well as combine different types of models is ensemble methods, which involve training several separate models and combining their predictions by training a new model [12]. This approach has been shown in a number of studies [12, 21] to produce superior results in comparison to standalone models. However, combining the final predictions of the models does not take into account that the static and temporal data are usually correlated, and more complicated models that for example concatenate the data before model training are susceptible to learning noise [20].

Hybrid models, a variant of ensemble models where the output of one model is the input of the next, can learn the correlation to a certain extent. Hybrid models have shown improved performance to ensemble models in certain cases [12], but are also susceptible to learning noise, and provide no interpretibility, as is the case with ensemble methods.

In recent years attention mechanisms have gained a lot of popularity and have been studied extensively in previous literature [22]. The concept of an attention mechanism is comparable to the human visual system, where humans focus on a small part of their total vision even though they are able to see much more. Similarly, attention mechanisms allow learning models to dynamically focus on specific features of the input, instead of allocating the same weights [23] to all features. Besides improvements in performance and computational efficiency, attention mechanisms have the ability to visualize which features are important in the prediction process, making the model

more transparent [22].

Furthermore, attention mechanisms have found application in the integration of various disciplines, such as image caption generation [24], visual question answering [25], and emotion recognition [26]. It has been used in the medical domain for tasks such as utilizing X-ray images and clinical reports for chest X-ray diagnosis [27], pancreas identification from medical images [28] and medical report generation [29]. Moreover, attention has made substantial contributions to natural language processing tasks including machine translation [30] and speech recognition [31]. It has also played a pivotal role in computer vision influencing image classification [32], image generation [33] and facial recognition [34].

Although hybrid and ensemble models can provide a way of combining different data while leveraging model strengths, the benefits of the attention mechanism are 2-fold. They provide context in the form of weights which improves model accuracy. These weights could also provide some form of interpretability by indicating which time steps are important.

This results in the following research question:

Main research question: Does an attention-based architecture which utilizes both static and time series data yield improved results to ensemble and hybrid architectures?

To answer the main research question we will investigate the following sub-research questions:

- Does integrating an attention mechanism into a model architecture result in higher accuracy and F1 scores, or lower mean squared error (MSE), compared to similar ensemble and hybrid architectures?

This leads to the following research objectives:

- To systematically compare the performance of models with and without attention mechanisms in terms of accuracy, F1 scores, and mean squared error (MSE), using standardized datasets to ensure consistency and reliability in the comparison.

- To analyze the results to determine if the addition of attention mechanisms consistently improves model outcomes over ensemble and hybrid models.
- Analyze whether key events within the data and structural patterns can be observed in the attention weights.

This has the potential to influence the design decisions of future machine learning systems, emphasizing model accuracy and efficiency. Furthermore, as attention mechanisms highlight the importance of specific features and timesteps, our study contributes to creating more transparent and interpretable models.

2. Background

Learning is to improve automatically with experience, according to T. Mitchell in his 1997 'Machine Learning' textbook [35]. With machine learning, computers learn complex patterns from past experiences by using techniques from statistics and mathematics. These patterns can be used to make predictions for unseen data. Prediction, in the context of machine learning, is estimating the unknown value of a system given the values of other measured features [36]. Prediction mainly falls into the category of machine learning known as supervised learning which is when a model learns by analysing 'labelled' data, where labelled refers to an input being tagged with a specific output. The model establishes a statistical relationship between the input and the output features [37].

Supervised learning can be broadly divided into two types, namely classification and regression [37]. Classification is used when the output variable is a category or a discrete value. For instance, determining whether an email is spam or not, or classifying images of cats and dogs [37]. Regression is used when the output variable is a continuous and numerical value, such as predicting a person's age or a stock's price [35]. Time series models, which will be discussed later in this section, are typically based on regression.

This section of the thesis describes the components and methodologies used in the experimental architecture, namely neural networks, recurrent neural networks, different types of ensemble modelling, hybrid models and attention mechanisms.

2.1 Deep learning models

Deep learning's core assumption is that data arises from composed features, potentially at different hierarchy levels. Other machine learning approaches make strong, task-specific assumptions such that the output at an unseen

data point should be approximately the same as the output at the nearest training point. By avoiding this, deep learning can generalize to a wider variety of problem structures [3]. Deep learning also has the advantage over traditional models that it can capture complex relationships and does not require careful feature selection [3].

2.1.1 Neural networks

An artificial neural network (ANN), usually called neural network (NN), is a computational model that is inspired by the structure and functional aspects of biological neural networks. Neural networks provide a robust approach to approximating real-valued, discrete-valued, and vector-valued target functions. A neural network consists of an interconnected group of artificial neurons divided into an input layer, a hidden layer, and an output layer. Each connection in the network is represented by a weight [1, 3].

The input layer receives raw data and passes it to subsequent layers. Hidden layers perform intricate computations by learning hierarchical features from the data. These layers progressively abstract and detect complex patterns. The number of hidden layers enables the neural network to capture and learn representations of increasing complexity. The output layer generates the final prediction by processing the information from the hidden layer through an activation function. An activation function is a non-linear mathematical function applied to introduce non-linearity, allowing the network to learn complex patterns. This layer-by-layer progression of transformations allows neural networks to model intricate relationships and make accurate predictions across various tasks [3, 19, 38].

The increased complexity may lead to overfitting on small datasets, and the choice of architecture, activation functions and regularization techniques significantly affect their performance [19]. Overfitting occurs when a model fits not only the underlying pattern of the data but also the random noise within the data set [1]. Deep neural networks, which involve training networks with numerous hidden layers, have significant performance advantages over plain neural networks and can prevent overfitting by

adding dropout layers after every layer [39]. Dropout layers are a form of regularization and remove a certain percentage of the neurons from a layer before its output is passed to the following layer.

2.1.2 Recurrent Neural networks

Recurrent Neural Networks (RNNs) are designed for sequence data, with connections that loop back on themselves to maintain a "memory" of previous inputs. This memory allows information from previous time steps to influence the current output which in turn enable RNNs to capture temporal dependencies in sequential data such as natural language processing and time series analysis [40].

The architecture consists of a sequence of cells, each processing one element of the input sequence and updating the hidden state based on both the current input and the accumulated information from previous inputs. This accumulated information is maintained in the hidden state, which is updated at each step using a consistent set of weights that apply throughout the sequence [3, 18]. However, it often suffers from the "vanishing gradient" problem [41], which hinders its ability to capture long-term dependencies. This issue led to the development of Long Short-Term Memory (LSTM) [42] architectures, which allow RNNs to retain information across longer sequences more effectively.

LSTMs achieve this through an architecture that includes three types of gates: input, forget, and output gates. These gates collectively decide which information should be remembered or forgotten as data flows through the sequence of the network, enabling the LSTM to maintain a longer memory [1]. This capability makes LSTMs highly effective for tasks that require the understanding of long-term dependencies in the data, such as complex language modelling, speech recognition, and time series forecasting

2.2 Ensemble & hybrid methods

In the following section, we consider existing methods of integration of static and time series data.

2.2.1 Ensemble models

"Ensemble methods are learning algorithms that construct a set of multiple individual classifiers (called base learners) and combine them to classify new data points by taking a weighted or unweighted vote of their predictions." [43]

Ensemble and hybrid models are two major approaches toward more accurate and reliable machine learning models [44, 45]. Ensemble models combine multiple models and thereby improve generalization by lowering the variance. Variance is defined as the variability of a model's predictions for a given data point [3]. High variance typically indicates that the model is sensitive to fluctuations in the training data, which can lead to overfitting. By combining multiple models in an ensemble, the individual variances of the models can be averaged out. This combination leverages the strengths of different models, each having a different perspective on the data because of different learning methods. Ensemble methods can be grouped into three categories, namely bagging [46], boosting [47] and stacking [48]. Hybrid models also leverage the strength of multiple models but differ from ensembles in how they integrate different types of models to create a single unified model [14].

Bagging (short for **b**ootstrap **agg**regation) combines multiple models of the same type, each trained on a different subset of data. The predictions of these models are aggregated to produce a final prediction. By combining models, bagging effectively averages the prediction, reducing the variance and improving generalization [46, 49]

While bagging is a parallel ensemble technique, boosting methods involve training a series of weak, typically the same type of models, sequentially. In boosting, each subsequent model is designed to correct the errors made by the

previous models, gradually improving the overall prediction performance [49]. Boosting is known for its high accuracy and ability to handle complex relationships in data. However, it can be prone to overfitting, especially on small datasets, due to its capacity to fit the noise in the training data [47]. Boosting is usually associated with Decision Trees but Drucker et al. [50] has already shown in 1993 that boosting applied to an ensemble of 3 neural networks can yield "dramatic" performance improvements compared to standalone models.

Stacking is when multiple base models are trained and their output is combined in a new model, known as a meta-classifier which is tasked with learning how to best combine the predictions from the base models and then make a final prediction [48]. Stacking involves training multiple base models, each tailored to utilize specific types of data, such as static data with static models and time series data with time series models. The outputs of these base models, in the case of deep learning models, are often not their final predictions but rather the raw output before being placed through an activation function. These outputs are combined and fed into a new model, known as a meta-classifier. The meta-classifier's task is to learn how to best combine these diverse outputs to make a final prediction. This approach not only allows the use of a variety of model types —including static and time series-based models— but also the models to be trained in parallel.

Ensemble methods, specifically stacking, allow a way of combining multiple models, thereby leveraging their different strengths, data suitability and reducing variance. In our experiments, we will create a stacked ensemble model which we refer to as a meta-classifier. This model will serve as a baseline model to which we compare our novel architecture, which will be discussed in Section 3.2.

2.2.2 Hybrid models

Hybrid models are similar to stacked ensembles in that they integrate different models [14], but differ in how data is integrated. Where stacked ensembles are trained independently and their output combined, hybrid

models are dependently linked such that the output of one model becomes the input of the next. This method allows the hybrid architecture to effectively learn the correlation between the learned representations of the static data and the raw time series data by training on both. See Figure 3.3 for a graphic regarding the model.

Models in a hybrid architecture leverage the data they were designed for, similar to stacking. However, as is the case with ensemble methods, appending the two datasets or outputs to one another can largely increase the dimensionality of the data on which the model must train, making it more susceptible to learning noise, which could impact performance [20]. Hybrid models do not strictly require a final meta-classifier, but cannot be trained in parallel. A common hybrid approach when working with static and temporal data is to train a neural network on the static data and concatenate its raw output, that is the output from its final hidden layer before the activation function, to the temporal data which is fed to the RNN, which then predicts the next sequence [51].

2.2.3 Concatenation

Initial time series models were tailored solely for time-series inputs, thereby missing out on the potential insights available within the static data, that could hold informative value [52]. Existing methods for integrating static inputs are to concatenate them at each time step (*static repeat*), at the first time step (*static first*) or last step (*static last*) of the time series data. One could argue that appending static features at every timestep (*static repeat*) equips the model with sufficient information to learn correlations between static features and each timestep. However, this method can lead to redundant computations since static features do not vary over time, potentially causing the model to learn noise present in the static data, which might degrade performance [20]. Lin et al. [13] propose an alternative by concatenating static data only at the last timestep (*static last*), a method that has shown improved outcomes compared to the *static repeat* approach. This is corroborated by findings that indicate both *static last* and *static first* reduce noise learning

issues inherent in the *static repeat* method [53].

However, regardless of where the static and time series data are concatenated, it increases the dimensionality of the data that the subsequent model components must train on, which leaves the model susceptible to noise in comparison to a static or time series-only model. In the next section, attention mechanisms will be discussed, and how they can be introduced to leverage the strength of different models whilst keeping dimensionality low.

2.3 Attention mechanism

Attention in machine learning is a mechanism that allows a model to focus on different parts of its input when producing an output. The concept of attention is best explained by comparing it to the human body's visual processing system which selectively focuses on important parts of an image while disregarding other non-essential details [24]. The attention mechanism was popularised in neural machine translation by Badanau et al. [30] to improve the performance of RNNs suffering from the vanishing gradient problem caused by very long sequences of data.

The attention mechanism helps the model to selectively focus on different parts of the input, giving more weight to the most relevant parts for a given task. This is achieved by comparing input elements against each other. Initially, a similarity score is computed for each input, typically by calculating the dot product. These raw scores undergo normalization using a softmax layer [3], resulting in weights between 0 and 1 that sum up to 1. These normalized weights are then used to create a weighted combination of the inputs, which form the context vector. Importantly, the entire process is differentiable which allows the model to adjust the attention weights as well as the model parameters during training through backpropagation [3], allowing the model to learn which parts of the input are pivotal for specific tasks as well as capturing non-adjacent relationships [30]. The mathematics of the attention mechanism are explained in Appendix 8.1. In addition, attention mechanisms can provide a level of interpretability to models, as the attention weights which show which part of the input the model is focusing

on to produce the output, can be visualized. This is especially useful in fields of computer vision [24, 54]. Attention also enables models to handle long sequences of data effectively, leading to better performance in tasks such as machine translation [30], image caption generation [24], speech recognition [31], visual question answering [25] and recommendation systems [55].

Various types of attention mechanisms exist and can be integrated into models in different ways. Usually, when working with RNNs, they are added as additional neural networks connected to the RNN [23]. In the well-known attention bidirectional RNN model for machine translation proposed by [30], the attention mechanism calculates the similarity between the hidden states of the encoder (RNN model) and the decoder (second RNN model), and the decoder then decides which part of the source sentence to pay attention to.

Attention has also been used in models which use different data types. In image captioning, in which the training data consists of image and textual data, Xu et al. [24] used an attention mechanism to allow an RNN model to focus on particular parts of the image when generating the next word in the caption. This resulted in state-of-the-art performance.

In the field of visual question-answering, models have been designed in which attention allows the model to align the words in a question with the sub-components of an information source such as a text [56] or an image [57]. The attention allows the model to focus on the relevant part of the data when producing an answer for a particular question, resulting in state-of-the-art performance and better model interpretability. In sentiment analysis, Zhu et al. [58] used cross-modal attention to dynamically determine the weight of the features of textual and image data when determining the sentiment polarity of each image-text pair. Zhang et al. [59] proposed an ensemble-based model utilizing a double attention mechanism for time series classification with heterogeneous features. Their model yielded competitive results whilst the attention mechanisms identified important features and improved interpretibility. In their paper, each feature is put through a low-level attention mechanism that learns which part of the feature is important. Then after each type of feature is trained on a suitable model, the high-level,

inter-feature attention between all features is learned, highlighting those that are of importance. In these multi-modal models, the attention mechanism allows the model to utilize one data source to aid the other, enhancing the overall predictive performance by selectively focusing on what is important.

2.4 Summary

To conclude, hybrid and ensemble methods effectively integrate static and time series models, capitalizing on different data types. This integration enables the models to learn correlations between the different types of data, enhancing the modelling of the problem and yielding more precise outcomes than those achieved by either static or time series models alone. Nonetheless, as discussed in Section 2.2.3, these methods may encounter challenges related to noise, stemming from the increased dimensionality caused by data concatenation.

In the following section, we introduce our novel architecture which uses an attention mechanism to combine the static and time series data. By introducing this architecture, we

1. leverage the strength of a static and a time-series model.
2. are able to use the attention mechanism to let the static data provide context to the time series data to let the model focus on which features are important, leading to an increase in performance.
3. can gain insight into which features are important in the prediction process by evaluating the attention weights.
4. are not concatenating static data, which means the dimensions of the data do not increase, which makes the model less susceptible to noise.

3. Method

As stated in the research question in Section 1, the purpose of this research is to evaluate whether an attention-based architecture which integrates static and temporal data can outperform ensemble and hybrid methodologies in terms of accuracy. We further inspect whether the attention weights align with key events within the data and whether we can observe structural dependencies. This section describes the type of experiments conducted, the architectures used in the experiments, the experimental setup, the validity of the experiments and how data analysis will be performed.

3.1 Classification and Forecasting

We evaluate both classification and forecasting problems. Models trained on classification problems output a single value, whereas models trained on forecasting problems output a sequence of values. The core of these architectural designs are the same but differ in how they produce an output. Figures 3.1, 3.2 and 3.3 show the model diagrams and the training is explained in sections 3.2 and 3.3.

For our model building, we use Tensorflow, a popular open-source deep learning library developed and maintained by the Google Brain Team [60]. Tensorflow has an ecosystem of tools and libraries to develop and deploy machine learning models. Its computational graph-based framework allows for efficient training of neural networks across a variety of platforms, making it a go-to solution for many machine learning developers. The RNN component in Tensorflow can output either the hidden layer of every timestep (shape (d_{RNN}, t)) or the hidden layer of the final timestep $(d_{RNN}, 1)$, where d_{RNN} is the number of hidden units per layer specified in the RNN and t is the number of timesteps. The NN component can produce an output of shape (d_{NN}, r) where d_{NN} is the number of hidden units in the NN layer, and r is the 2nd dimension of the input to the layer, which is preserved in

its output. This means that if the input to the NN layer has shape (d_{RNN}, t), the output will have shape (d_{NN}, t), and if the input has shape $(d_{RNN}, 1)$, the output will have shape $(d_{NN}, 1)$.

For classification, the final output shape must be $(c, 1)$, where c is the number of output classes. Each output represents the independent probability of belonging to each class, which is achieved using a sigmoid activation function, suitable for binary classification scenarios. For forecasting, the shape must be $(1, t)$. To achieve this, the classification design has a 2nd RNN (decoder) which outputs shape $(d_{RNN}, 1)$, which is fed into a shallow NN outputting a shape of $(d, 1)$. For forecasting, the decoder RNN outputs a shape of (d, t) which is fed into a shallow NN outputting a shape of $(1, t)$.

3.2 Attention based architecture

The model (Figure 3.1) developed for this study was designed to enhance performance and solve the dimensionality problem of the standard methods at the end of Section 2. It is an integrated deep-learning model that processes static and temporal data to be used for either forecasting or classification. It uses an attention mechanism to calculate a set of weights which are based on the similarity of the static and time series data.

The training of the attention model is similar to that of standard deep learning models [1], and is described as follows:

1. **Initialization:** The weights of the NN, RNN, attention mechanism, decoder RNN and shallow NN are randomly initialized.
2. **Forward pass during training**:
    - The static data is processed through the NN once and the hidden states (Q) are calculated. The output shape is $(d_{NN}, 1)$
    - Similarly, the time series data is processed through the RNN once and its hidden states are calculated. The output shape is (d_{RNN}, t).
    - The attention weights are calculated by taking the softmax of

the dot product of the hidden states from the NN (Q) and every hidden state corresponding to every timestep in the RNN (K). Every weight represents the importance of a timestep in the context of the static data.

- The attention weight multiplied by the hidden states of the RNN (V) form the context vector. See Appendix 8.1 for a detailed explanation of the mathematics behind the attention mechanism.
- The context vector is fed into the decoder RNN which outputs either a single hidden layer, for classification problems, or a sequence of hidden layers, for forecasting. This output is fed to the shallow neural network, which makes the final prediction.

3. **Compute Loss**: For regression tasks, the loss is computed by calculating the Mean Squared Error (MSE) between the prediction and the actual target variable. For classification tasks, binary cross-entropy is used to calculate the loss.
4. **Backpropogation**: The gradients of the loss with respect to each weight in the model are calculated. The weights are updated in the direction which minimizes the loss. This includes the weights of the NN, the RNN, the attention mechanism, the decoder RNN and the shallow NN. The initial size of these updates is constrained by using a learning rate which is adjusted during training using an Adam optimizer [61].
5. **Validation**: After each pass (forward step, compute loss and backpropagation) the model is used to make predictions on a validation dataset. This is a portion of the original dataset which is not used to update the model but to monitor how well it is performing on unseen data during training. When the validation loss starts increasing whilst the training loss continues decreasing, it means the model is overfitting and training can be halted.

To summarize, after initialization, the models are trained by feeding the data through it, calculating the loss and updating the weights. The process of calculating the loss and updating the weights refines the weights to reduce

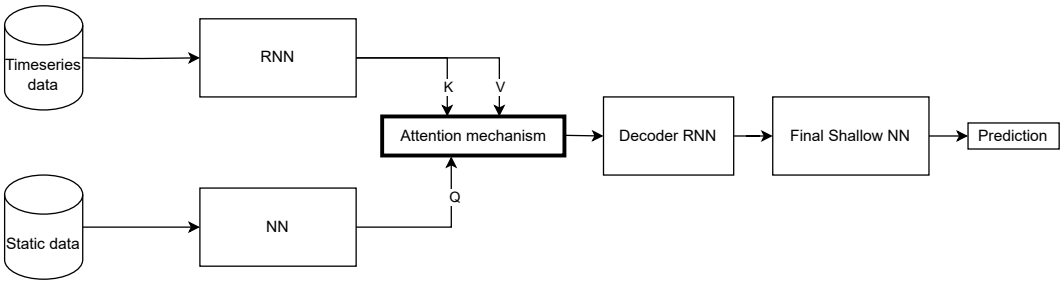

Figure 3.1: The attention architecture

the loss. This means that over multiple iterations, the attention mechanism will have been adjusted to focus on the most relevant time steps, as guided by the training data and the loss function.

The training is repeated until the mean squared validation error increases or in the case of classification problems, the validation accuracy no longer increases, or if the maximum number of forward passes (epochs) is reached.

3.3 Approach

To evaluate this attention architecture, we compare it to several architectures listed below. As stated in Section 2, every architecture has a unique strength and a different method of learning the correlation between data types. All of them, however, face a challenge concerning large dimensionality and have no interpretability.

With our introduction of an attention architecture, we present a new way of learning relationships between different data types. By comparing it to established methods, we evaluate whether the attention mechanism is indeed able to learn the correlation between data types, whether it can achieve similar performance as well as offer insight as to which features are important during the prediction process.

Below we present the three models/architectures to be used in our experiments:

1. **An Ensemble consisting of a static (NN) and a time series model (RNN). Also referred to as meta classifier** In the ensemble approach each model independently trains on the specific type of data. The raw

output of the NN is concatenated to the RNN's first hidden state which in turn is fed into the decoder RNN. The output of the decoder is fed into a shallow NN, which is used to make the final prediction. (Figure 3.2)

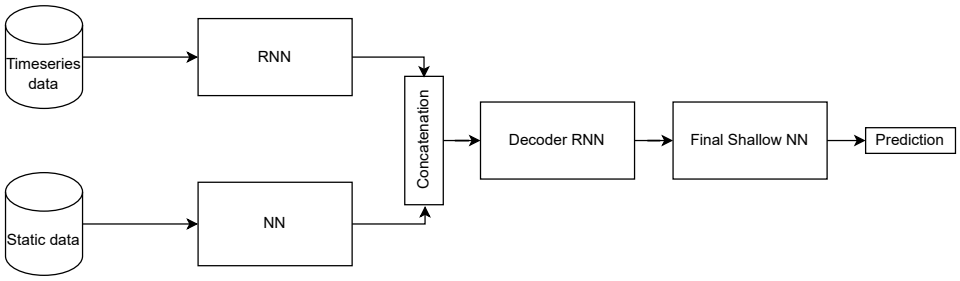

Figure 3.2: Meta classifier model

2. **Hybrid model consisting of a static (NN) and a time series model (RNN).** In the hybrid approach, the static model is trained and its raw output is concatenated to the 1st time step of the raw time series data which is then fed into the RNN model. The output of the RNN is fed into the decoder RNN, and its output is fed into the shallow NN. (Figure 3.3)

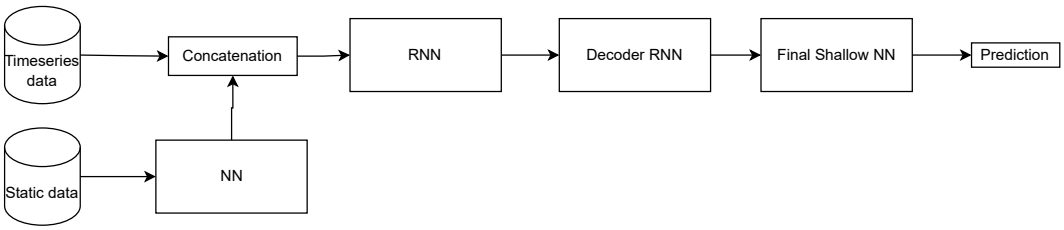

Figure 3.3: Hybrid model

3. **Novel attention architecture consisting of a static (NN) and time series model (RNN).** In this architecture (described Section 3.2) the static and time series models are trained independently. The raw outputs of both models are combined in an attention mechanism. The output of the attention mechanism is fed into the decoder RNN which subsequently flows into a shallow NN, which is used to make the final prediction. (Figure 3.1)

3.3.1 Nuances in the concatenation block

In the meta and hybrid architectures, the static and time series data can be concatenated in different manners. We consider 3 different methods of concatenation, namely no sequence (simple), first step and every step concatenation. Previous works have shown that first or last-step concatenation performs better than every-step concatenation, however, it is unclear whether this also holds for smaller datasets.

3.3.1.1 Meta classifier

No Sequence: This method only occurs in the meta-classifier model. It has a single RNN component which outputs the hidden layer of the final timestep, shape $(d, 1)$, which is appended to the output of the NN to have shape $(2d, 1)$ before being fed into the final shallow NN (Figure 3.4). This architecture is simpler than the other architectures due to the absence of the decoder and therefore has fewer parameters to train. It is less likely to overfit in classification tasks but more prone to underfitting in forecasting problems.

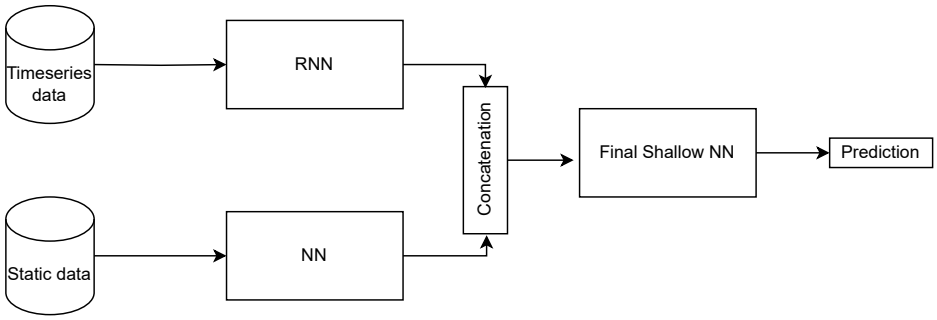

Figure 3.4: Meta classifier no-sequence model

1st step concatenation: In this method, the decoder RNN from Figure 3.2 outputs the hidden state at every timestep. Its output is of shape (d, t) where t is the number of timesteps and d is the dimension of each hidden state. The raw output from the NN, which has shape $(d, 1)$ is concatenated to the first timestep (Figure 3.5), whilst the remaining timesteps are padded with -1 to ensure the shape of the matrix is $(2d, t)$ before being fed to the decoder RNN component, which outputs the hidden state of the final timestep, shape

$(d, 1)$, for classification tasks and (d, t) for forecasting tasks, before going into the final shallow NN, similar to the 'No sequences' model.

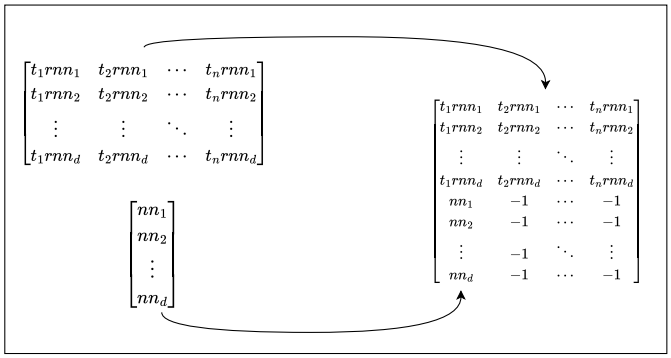

Figure 3.5: 1st step concatenation block Meta classifier architecture

Every step concatenation: This architecture is very similar to '1st step concatenation', except the output from the NN is concatenated to every timestep of the RNN output (Figure 3.6). All shapes are the same as the previous architecture, except that this architecture does not require padding.

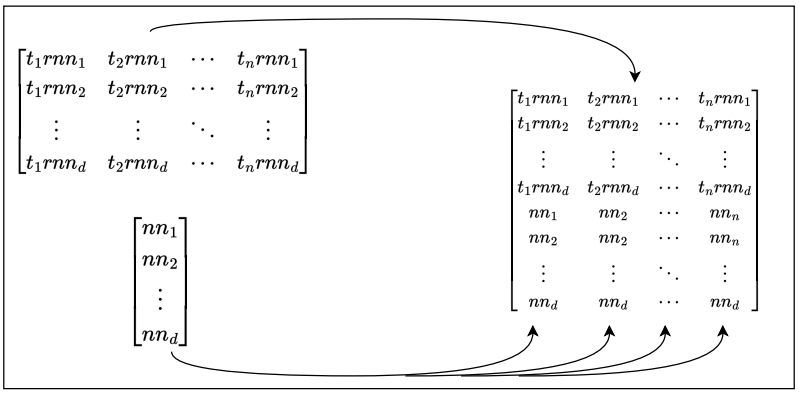

Figure 3.6: Every step concatenation block Meta classifier architecture

3.3.1.2 Hybrid architectures

In the hybrid architecture (Figure 3.3) the output of the NN (shape: $(d, 1)$) is concatenated to the time series data (shape: (X_t, t), where X_t is the number of time series features), which is then fed into the RNN. Here we again apply 1st step (Figure 3.5) and every step (Figure 3.6) concatenation (resulting shape: $(X_t + d, t)$). The output from the RNN is (d, t), which is fed into the

decoder RNN. The decoder, similar to the meta every step and meta 1st step concatenation architectures, outputs the hidden layer of the final timestep, shape $(d, 1)$, for classification tasks and (d, t) for forecasting tasks, before going into the final shallow NN.

3.4 Experimental settings

Hyperparameters : The same hyperparameter search was conducted for each experiment. For each architecture, we varied the initial learning rate from 10^{-3} to 10^{-6} whilst keeping the number of units fixed at 128 and the number of NN layers at 7. Next, we varied the number of layers in the NN from 2 to 7 whilst keeping the number of units fixed at 128 and using the learning rate which resulted in the highest performance. Next, we varied the number of hidden units on which to train from 16 to 128 in increments of 2^n. To save time, the batch size, type of optimizer, weight initializers, number of RNN layers, and dropout ratio after each layer in the neural network were kept constant across each architecture, as shown in Table 3.1.

Hyperparameter	**Values**
Number of layers in NN	2, 3, 4, 5, 6, 7
Number of units from layer 1 to n for NN	16, 32, 64, 128
Number of units RNN	Equal to hidden units of NN
Activation function in NN and RNN	ReLu
Number of units final shallow NN	Classification = 2, else 1
Number of layers final shallow NN	1
Weight initializers	RandomUniform$(-0.2, 0.2)$
Activation function in final shallow NN	Classification = sigmoid, else ReLu
Initial learning rate	10^{-3}, 10^{-4} , 10^{-5}, 10^{-6}
Optimizer	Adam Optimizer
Max epochs	900
Early stopping	True, after 100 epochs
Class weights	Classification = True, else False
Percentage Dropout	20% of units dropped after every NN layer
Percentage of majority class to exclude	0%
Batch size	1
Loss function	Classification = binary_crossentropy, else MSE

Table 3.1: Hyperparameter search settings

We used a single-layer Bi-directional LSTM for our RNN with the number

of units equal to the number of units in the NN. For the decoder RNN, we use a single-layer LSTM with the number of units equal to the number of units in the NN.

Model Training and Evaluation:

Similar to Section 3.2, the above models/architectures are trained, validated and tested on a training, validation and test set. After each training epoch, the model is tested on the validation set to ensure it is not overfitting. Once the model has reached the maximum number of training epochs (900) or the validation accuracy is no longer increasing for 50 epochs, training is halted, the weights for the model at the epoch resulting in the highest accuracy are returned, and the model is tested on the test dataset. If the 'Early stopping' parameter were set to false, the architecture would continue to train for the entire 900 epochs, resulting in overfitting; the architecture yields a high training accuracy but a low test accuracy. The training time has also increased significantly.

When dealing with classification tasks, model accuracy does not always best reflect the architecture performance, especially if the data is unbalanced, that is having more cases belonging to one class than the other, as the model can achieve a high accuracy by simply predicting the majority class. Therefore we report also the precision, recall and F1 scores, which are described below.

Precision measures the accuracy of the positive predictions made by a model. It is calculated as the ratio of true positives (TP) to the sum of true positives and false positives (FP), where a true positive is an outcome correctly identified as positive, and a false positive is an outcome incorrectly identified as positive.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.1}$$

Recall measures the ability of a model to identify all relevant cases within a dataset. It is calculated as the ratio of true positives to the sum of true

positives and false negatives (FN), where a false negative is an outcome incorrectly identified as negative.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.2}$$

The F1 score is a harmonic mean of precision and recall, providing a metric that balances both the precision and the recall of a model. It is particularly useful for evaluating performance on a dataset with an imbalance in class distribution. The F1 score reaches its best value at 1 and its worst at 0.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.3}$$

The evaluation metrics used include accuracy through F1 score for classification problems, Mean Squared Error (MSE) for forecasting problems and training time (number of epochs).

3.4.1 Attention weights

The attention mechanism within our architecture generates a set of weights that identify which timesteps are considered most important when forecasting a sequence or making a classification prediction. Visualizing these attention weights can also reveal aspects of the dataset's structure. In our experiments, we will plot a known critical event, such as patient medication usage, over attention weights, to determine if they align. If alignment is observed, it suggests that the attention mechanism has successfully learned this important feature of the dataset

3.5 Validity

To ensure validity and reliability we discuss and conform to the criteria described by Yin in his work "Case Study Research Design and Methods" [62] to judge the quality of our research design. The criteria are construct

validity, internal validity, external validity and reliability.

To ensure the **construct validity**, the architecture is compared to existing architectures. The goal of our research is to determine whether our attention architecture can provide accurate next value predictions for a time series. To measure the accuracy, the architecture's ability to generalize is compared to that of existing architectures employing methods of data integration. It is known that more data typically allows for more accurate models. However, if the attention architecture performs poorly, it could be that the architecture is unable to learn the correlation between the static and temporal data. This can be a result of either the architecture not being suitable, or the lack of data not allowing the model to learn sufficient relationships.

To ensure the **internal validity**, the experiment is validated using a test dataset (Section 3.4). The mean and standard deviation of each performance metric are computed for every architecture, on which a data analysis (Section 3.6) is performed.

To ensure **external validity**, the proposed experiment is performed on three different datasets in three different domains. The datasets contain static and time series data. The results will vary per domain, but overall the same conclusions should be reached.

To ensure the **reliability** of our research, our code for each model in every architecture will be documented and made available. Due to privacy regulations, the data cannot be publicly shared, however, the dataset will be frozen on the UMCU server. The Sepsis dataset is publicly available [63] and the Fibonacci dataset will be shared on GitHub.

3.6 Data analysis

As mentioned, the performance of each architecture is evaluated by calculating the MSE, accuracy and F1 score. To ensure validity the experiment is repeated 15 times, and the mean and standard deviation of each performance metric is recorded. To determine if the differences in performance between architectures are statistically significant, we perform a

Kruskal-Wallis test at the group level. This is followed by an epsilon-squared calculation to assess the effect size. For post hoc analyses, depending on the data distribution, we either use an independent samples t-test (for Gaussian data) with Cohen's d to measure effect size, or a non-parametric Dunn's test with a Bonferroni correction to adjust for multiple comparisons, accompanied by Cliff's delta to quantify the effect size.

4. Data

In this section, we consider the four datasets used in the experiments. For each dataset we provide its context, describe the variables used, how the preprocessing was done and how the experiments are set up. Supplementary information about the datasets can be found in Appendix sections 8.3 and 8.5.1.

4.1 Cystic fibrosis

4.1.1 Background

Cystic fibrosis, an inherited disorder that limits a person's lifespan, is caused by an autosomal recessive gene mutation [64]. This genetic anomaly affects a gene known as the cystic fibrosis transmembrane conductance regulator (CFTR), which plays a critical role in ion transport and the cleaning mechanism of the airways, referred to as mucociliary clearance. When the CFTR gene isn't functioning correctly, it can result in various organ malfunctions. One significant consequence is a persistent airway infection, leading to ongoing lung damage. This ultimately results in respiratory failure, which is often the cause of premature death in cystic fibrosis patients.

Over the years various medications have been introduced to improve the quality of life of patients. Double modulators have yielded limited success, showing only temporary improvements in a patient's pulmonary function [65]. However, since 2020, a triple modulator has been approved for use in the EU and has shown significant results. This triple modulator therapy enhances the function of the CFTR protein more comprehensively than previous treatments by correcting the protein defect at multiple stages of protein processing and function. As a result, patients have experienced marked improvements in pulmonary/lung function, nutritional status, a reduction in sweat concentration and pulmonary exacerbations [66, 67].

These enhancements have not only prolonged life expectancy but also significantly improved the quality of life for many individuals with cystic fibrosis.

The dataset used for our experiment consists of patients receiving treatment for cystic fibrosis (CF) from the Wilhelmina Children's Hospital (WKZ) which is part of the larger academic hospital, University Medical Center Utrecht (UMCU). This medical centre has more than 11.000 employees and is the largest public health institute in the Netherlands. Their CF department is the largest in the country and has a research program aimed at developing personalized medicine for patients[1]. An advantage of collecting data from a large hospital is that it generally has more patients and thus more data than smaller clinics. Patients in this dataset have differing ages, medical histories and medical conditions, but have all been taking the triple CFTR modulator medication known as Elexacaftor–Tezacaftor–Ivacaftor (Kaftrio) since January 2022. Patients visit the hospital once every 3 - 6 months, where several measurements are taken.

The data consists of static and temporal features of about 300 patients. This may seem like a small amount but considering that cystic fibrosis is a very rare disease affecting 162 428 people worldwide, this is a considerable number of patients [68]. The static features do not change over time such as sex, date of birth and type of cystic fibrosis mutation. The temporal (time series) data is described as varying, with some patients having more than 50 measurements and others less than 5. The variables are measured periodically, namely lung function and type of medication used.

4.1.2 Goal of experiment

The dataset is used for two experiments, namely forecasting and classification. For each experiment, the dataset is tailored in a slightly different way. For the forecasting experiment, the goal is to use the measurements a patient has before Kaftrio use to predict what their percentage predicted Forced

[1] https://www.hetwkz.nl/nl/ziekte/cystic-fibrosis

Expiratory Volumne in 1 second (ppFEV1) over time will be after Kaftrio use.

The classification experiment is similar to the forecasting experiment in that it uses patient measurements before they start taking Kaftrio. However, its goal is to predict whether a patient's ppFEV1 will improve, framing it as a binary classification problem. This is a less complicated problem to solve than forecasting the exact ppFEV1 value over time, as the model essentially only has to determine whether or not the ppFEV1 will be higher or lower.

The remainder of the section is structured as follows: The variables of the dataset are presented, followed by an explanation regarding the variable preprocessing, patient extraction criteria and a subsection describing the experiments. A snapshot of the dataset as well as further variable description and alignment are presented in Appendix 8.3.

4.1.3 Variables

Description of data

A description of the specific columns and how they were extracted can be found in Table 4.1. The columns, measurements, were recorded at different points in time, and extracted from different tables, as described in Table 4.3. The final static and time series datasets come from the same source, the difference being that a single row in the static dataset represents a patient in terms of their measurements before their first CFTR modulator use. Every patient in the static dataset only occurs once. A single row in the time series dataset represents a 'snapshot' of a patient in terms of their measurements from their first CFTR use to their last ppFEV1 measurement. A patient can occur multiple times in the time series dataset, each new row representing a patient measurement at a different point in time. Together, all patients' measurements form a sequence. Tables 8.4 and 8.5 in Appendix 8.3 show synthesized extracts of the static and time series data used in the experiments before scaling has been applied. Figure 4.1 shows the correlations between the variables.

Variable name	Extraction description	Transformation description	Variable description	Dataset
Continuous				
Height	Extracted using regex expressions.	Min-Max Normalized	Patient's height in cm.	Static
Weight	Extracted using regex expressions.	Min-Max Normalized	Patient's weight in kg.	Static, Timeseries
Sweat-concentration	Filtered lab results on "Chloride-ZW"	Min-Max Normalized	Last sweat-concentration measurement before CFTR use.	Static
ppFEV1	Calculated	Min-Max Normalized	percent predicted Forced Expiratory Volume in 1 second. Ratio of CF patient FEV1 to standard FEV1 based on age, sex, height, and ethnicity.	Static, Timeseries
Date		Calculated elapsed time between measurements	Date on which measurement was taken.	Timeseries
Binary				
Diabetes	Filtered for keyword 'diabetes'		Indicates if the patient had diabetes before CFTR use.	Static
Sex				Static
CFTR usage		Modulators were converted to binary columns	Indicates CFTR modulator use	Timeseries
Anti-biotics usage (Nebulized)	Filtered for keywords 'Amikacine', 'Aztreonam', 'Ceftazidim', 'Colistine', 'Meropenem', 'Tobramycine'		Indicates if the patient had been using anti-biotics in the past 30 days	Static, Timeseries
Anti-biotic usage (Oral)	'Azitromycine', 'cotrimoxacol', 'flucloxacilline', 'levofloxacine', 'amoxicilline', 'ciprofloxaciline'			Static, Timeseries
Discrete				
Year of birth		Min-Max Normalized		Static
Number of admissions	Aggregated hospital visits where keyword 'spoed' is present or column spoed = 1		Number of emergency hospital visits before CFTR use.	Static
has F508del	Whether the patient has the F508del gene mutation	F508 encoded as 1	Indication of patient gene mutation	Static

Table 4.1: Column extraction and transformation descriptions for static and time series datasets

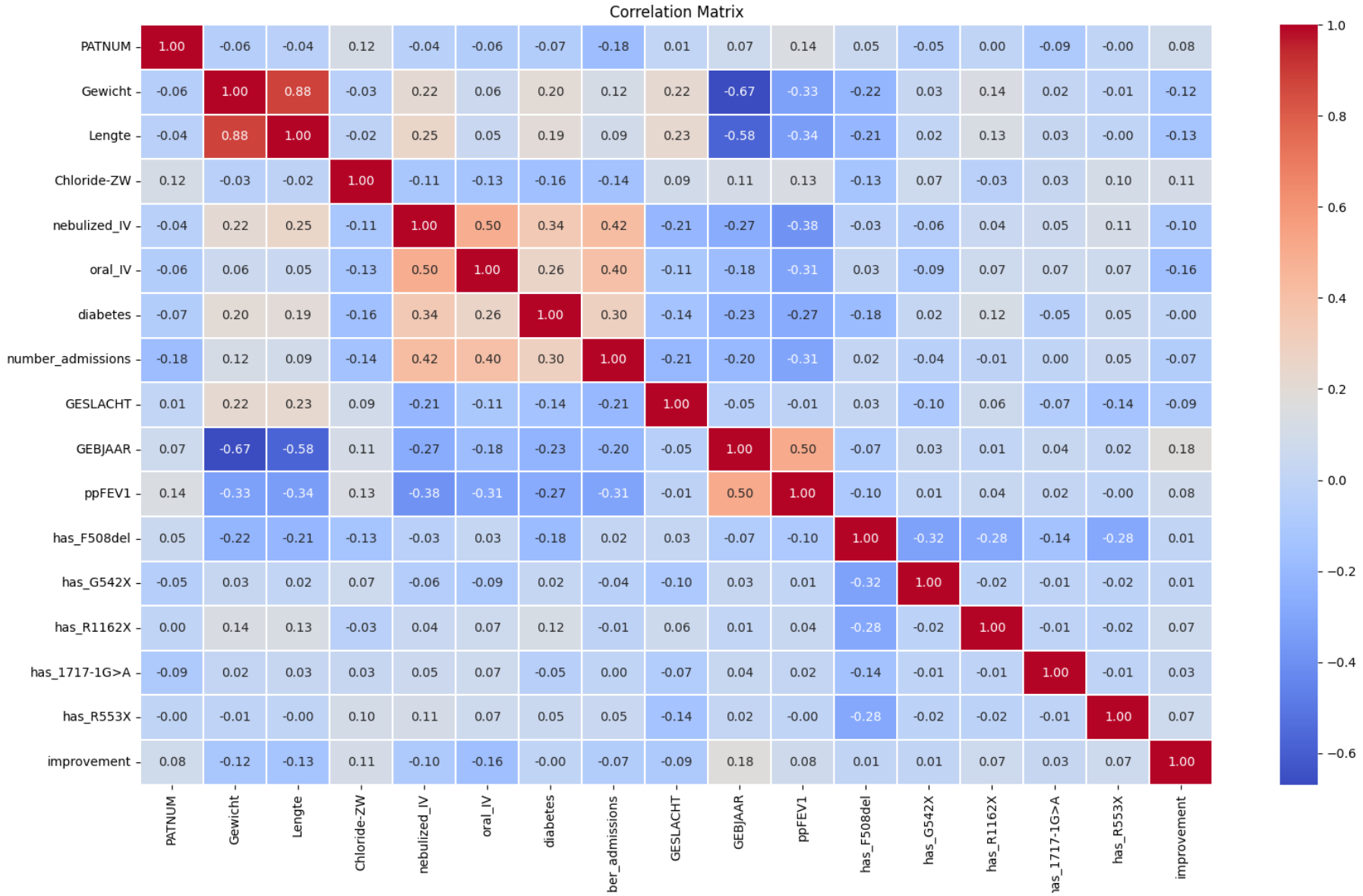

Figure 4.1: Correlation coefficient matrix for the CF data

4.1.4 Preprocessing

The data processing aims to convert the data into a format that is suitable for model building. It involves extracting data from different tables, integrating and aligning the data appropriately, addressing missing values, removing outliers, scaling the data and padding/truncating sequences where necessary.

Data Extraction Criteria

Table 4.3 describes the different columns used and from which tables they were extracted. Furthermore, patients were only included in the dataset if they met the following criteria:

1. Patients have used a CFTR modulator and are therefore present in the *Apotheek* table.
2. Patients have a sex and year of birth value and are therefore present in the *Demo* table.
3. Patients have a FEV1 (lung function) measurement at least before their first CFTR use date and are therefore present in the *Longfunctie* table
4. Patients have a height and weight measurement at least before their first CFTR use date and are therefore present in the *Metingen* table
5. Patients have a sweat concentration value at least before their first CFTR use date and are therefore present in the *Lab* table.

To calculate the ppFEV1 of a patient, their height, age, sex, FEV1 and ethnicity is needed as described in Equation 8.2. This implies that a patient is excluded if any of these measurements were missing or in the case of height, could not be imputed.

Missing values As described above, if a patient does not have all of these variables before the 1st CFTR modulator use date, then the row is excluded. This is true for the static and time series data. Certain missing values were deduced from the data, such as gene mutation, height and weight. For gene mutation, after consulting a domain expert, it was assumed that a patient

has *F508del* as a second gene allele if this value is missing in the dataset. *F508del* is the most common gene mutation in cystic fibrosis patients [66]. Of the other 27 mutations present in the dataset, we included only the 4 most occurring mutations, namely *G542X*, *R1162X*, *1717-1G>A* and *R553X*. In the *gene mutation* table 80% of the patients had *F508del*. *R1162X*, the 2nd most occurring mutation, was present in less than 3% of the patients. These mutations were represented as binary columns to indicate their presence.

In cases where the height or weight measurements were missing, the average of the patient's preceding and succeeding height and weight measurements were used, given that the missing measurement's date still falls within 180 days of the preceding and succeeding value. A window of 180 days was decided as in general a patient's height and weight do not change significantly during this time.

Outliers Outliers skew the distribution of the data [69] and can cause the model to overfit. In certain evaluation metrics, such as Mean Squared Error and Sum of Squares error, outliers can significantly decrease model accuracy and were therefore removed from the dataset. They were identified using the using the interquartile range (IQR) [70]. IQR is the range between the first and third quartiles $IQR = Q3 - Q1$. The lower limit is defined as $Q1 - 1.5IQR$, and the upper limit as $Q3 + 1.5IQR$. The region between the upper and lower limit contains 99.3% of the data. In each column, a value that falls below the lower limit or above the upper limit is considered an outlier and the entire row is removed from the dataset.

Input and output sequences The time series data included in the dataset ranges from the patient's first CFTR usage until their last lung function measurement. The time series data is split into 2 sequences, namely pre- and post-Kaftrio. The pre-Kaftrio sequence includes all measurements from the first CFTR usage up until the last lung function measurement before the patient started using Kaftrio. The post-Kaftrio sequence only includes the ppFEV1 measurements of a patient since the patient started using Kaftrio up until their last ppFEV1 measurement. The pre-Kaftrio

sequence is also referred to as the input sequence and the post-Kaftrio as the output sequence. The forecasting experiment is tasked with predicting the post-Kaftrio sequence.

The classification experiment does not require the output sequences. Instead, it predicts whether or not a patient's ppFEV1 will improve after taking Kaftrio. A patient is labelled as improving if their average ppFEV1 value after taking Kaftrio is higher than their last ppFEV1 value before Kaftrio. This average is calculated by summing together a patient's output sequence and dividing by the length of the sequence.

Padding and Truncating Patients in the dataset visit the hospital at different times and at different frequencies. At each hospital visit, measurements are taken. Each patient's measurements are treated as a sequence which is fed into the RNN component. The forecasting experiment design requires all sequences to have the same length. Input sequences shorter than 5, the mean length of the output sequences, were padded with -1 at the start of the sequence until they were length 5. A masking layer was added to ignore weight updates for these padding values. Output sequences were padded by repeating the last value in the sequence until it was length 5. Input sequences longer than 5 were truncated to only include the last 5 measurements in the sequence and output sequences were truncated to include the first 5 measurements.

The input sequences in the classification experiment are set to length 20. Since the classification experiment does not require the output sequences, the input sequences can be any length, however longer input sequences do lead to longer training times, therefore sequences were truncated to the last 20 measurements. Figures 4.2 and 4.3 show the sequence lengths of all patients.

Integrating and alignment An important part of data processing is ensuring the dates of the variables are aligned. For the **static data**, the most recent values of the variables immediately before a patient's initial usage of CFTR modulators are used.

All **timeseries data** have dates between the patient's first CFTR use and

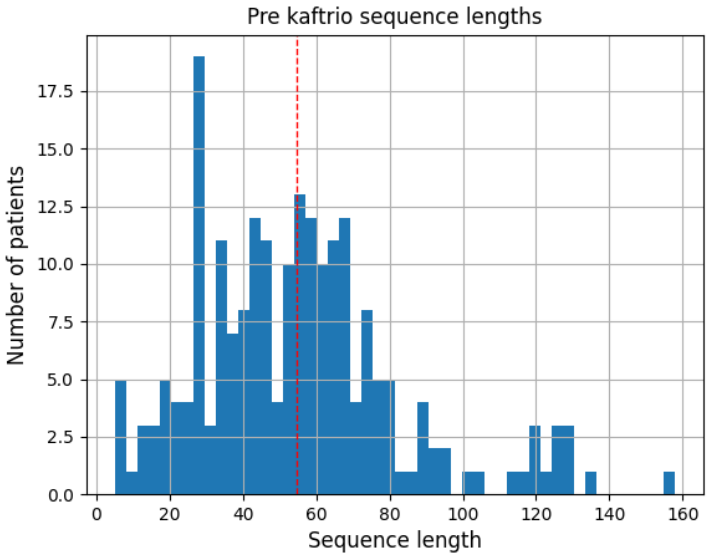

Figure 4.2: Pre-Kaftrio sequence lengths. The mean is indicated by the red line.

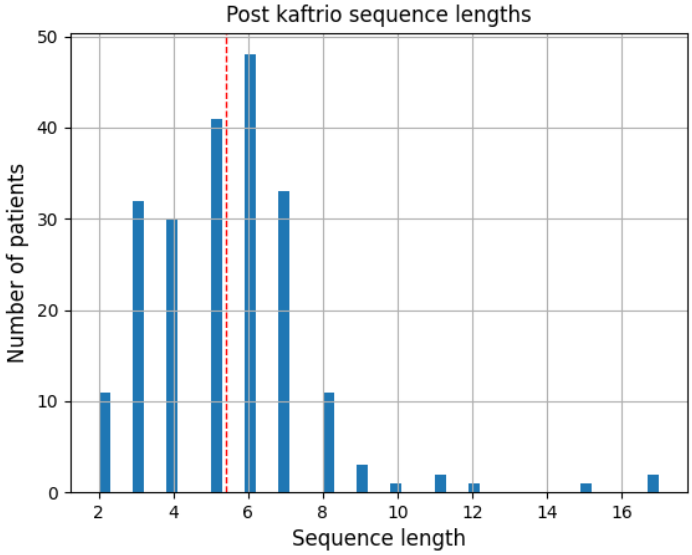

Figure 4.3: Post-Kaftrio sequence lengths. The mean is indicated by the red line.

their last lung function measurement date. However, different types of measurements, ex. height and lung function, are often not taken on the same date, meaning that the dates are not aligned. Certain columns do not indicate a measurement but rather a change in usage or diagnosis. For example, the CFTR modulator usage in the *Apotheek* table shows only the start and stop dates of the modulator, and the *Medication* table indicates when a patient

was using anti-biotics, but these dates do not align with lung function dates.

Since ppFEV1 is the target variable, the dates of when a FEV1 measurement takes place represent the timestamps which are fed into the model. This means that all other measurements are aligned with these dates. When measuring FEV1, physicians take multiple readings on a single day and view the highest FEV1 of a patient as their actual measurement. The same is done for the dataset; if multiple FEV1 measurements occur on a given day, the highest is kept, and the rest are discarded.

Scaling To ensure that features are on a uniform scale, preventing variables with larger magnitudes from dominating the learning process in machine learning models, variables are scaled to be within the range 0 and 1 [1]. This is done by subtracting the minimum of the column from each data point and then dividing it by the range, maximum - minimum. This ensures that all features initially contribute equally to the learning process. All discrete and continuous variables, as can be seen in Table 4.1, were scaled.

Dataset size Table 4.2 show the size of the training test and validation sets used in the CF experiment.

Dataset	Forecasting	Classification	Improvement	No-Improvement
Training set	164	153	122 (80%)	31 (20%)
Test set	33	34	25 (74%)	9 (26%)
Validation set	19	37	29 (78%)	8 (22%)
Total	216	224	176 (79%)	48 (21%)

Table 4.2: CF Dataset Sizes

4.1.5 Experiments

Regarding the CF dataset, we consider both forecasting and classification problems. For forecasting, the goal is to predict post-Kaftrio ppFEV1 measurements over time using the static and time series pre-Kaftrio measurements of a patient. This approach bears a practical benefit, as we can forecast how a patient's ppFEV1 will be impacted by the use of Kaftrio, based on their medical history. As mentioned in Section 3.1, in forecasting

Table name	Description	Columns needed
apotheek	Patient CFTR usage.	CFTR type, Start date, Stop date
metingen	Diverse measurements	Height, weight
diagnose	Diagnoses data	Diabetes
demo	Demographic data	Gender, Year of birth
Medicatie	Patient medication usage	Anti-biotics usage
opnames	Recorded patient hospital visits	Number of Emergency hospital visits
lab	Results of diverse medical tests	Sweat-concentration
longfunctie	Patients lung function measurements	FEV1, ppFEV1
gene mutation	Diverse patient properties and measurements	mutation allele2

Table 4.3: Raw data table description

problems the decoder RNN component outputs a vector for every timestep of the sequence, resulting in a shape of (d, t). This output is fed to a shallow NN with a single hidden unit, tasked with reducing the dimensionality to $(1, t)$ in order to predict the final sequence of predictions.

For classification, the goal is to predict whether or not the average post-Kaftrio ppFEV1 measurement is higher or lower than the last pre-Kaftrio ppFEV1 value. If this average is higher, it is labelled as an improvement, else it is not. This is a binary classification problem, in which the decoder RNN component outputs a single vector of shape $(d, 1)$ which is fed into a shallow NN, having a number of hidden units equal to the number of target classes, which in this case is 2. For both the forecasting and classification problem we will test the difference in performance between the attention, hybrid_1_step, hybrid_every_step, meta_1_step, meta_every_step architectures, explained in Section 3.3.

4.2 Sepsis

4.2.1 Background & Problem statement

The sepsis dataset [63] is a publicly available event log containing 1050 cases of patients who were admitted to a Dutch hospital over 1.5 years because of a sepsis condition. Sepsis is a life-threatening condition that arises when the body's response to an infection causes injury to its tissues and organs, potentially leading to organ failure and death if not promptly and effectively treated [71]. The original purpose of the dataset was to train a process model of the patient trajectories using process mining techniques. Each case consists of a sequence of activities the patient underwent before being released from the hospital. The authors also investigated the trajectory of patients that return to the emergency room (ER) [72]. We train our models to predict whether or not a patient will return to ER.

4.2.2 Variables

Table 4.4 shows the different variables in the dataset. The majority of the dataset contains binary variables, with age and activity being the only exceptions. After converting the activity column to static variables, this column housed the 'Return ER' target variable. Figure 4.4 shows the class distribution of the number of patients returning to the ER.

Column Names	Datatype	Dataset	Reason excluded from training
case_id	Binary	Static	Random
InfectionSuspected	Binary	Static	high linear correlation
DiagnosticBlood	Binary	Static	high linear correlation
DisfuncOrg	Binary	Static	
SIRSCritTachypnea	Binary	Static	
SIRSCritHeartRate	Binary	Static	
Infusion	Binary	Static	high linear correlation
DiagnosticArtAstrup	Binary	Static	
Age	Discrete	Static	
DiagnosticIC	Binary	Static	high linear correlation
SIRSCriteria2OrMore	Binary	Static	high linear correlation
DiagnosticXthorax	Binary	Static	high linear correlation
SIRSCritTemperature	Binary	Static	
DiagnosticUrinaryCulture	Binary	Static	high linear correlation
SIRSCritLeucos	Binary	Static	
DiagnosticLacticAcid	Binary	Static	high linear correlation
Hypoxie	Binary	Static	
DiagnosticUrinarySediment	Binary	Static	
DiagnosticECG	Binary	Static	
IV Liquid	Binary	Static	
IV Antibiotics	Binary	Static	
Admission IC	Binary	Static	
ReleaseType	Categorical	Static	
timestamp	Datetime	Timeseries	
Leucocytes	Continuous	Timeseries	
CRP	Continuous	Timeseries	missing values
LacticAcid	Continuous	Timeseries	missing values
Return ER	Binary	Target	

Table 4.4: Column Names, Datatypes, Dataset Classification, and Training Exclusions of Sepsis data

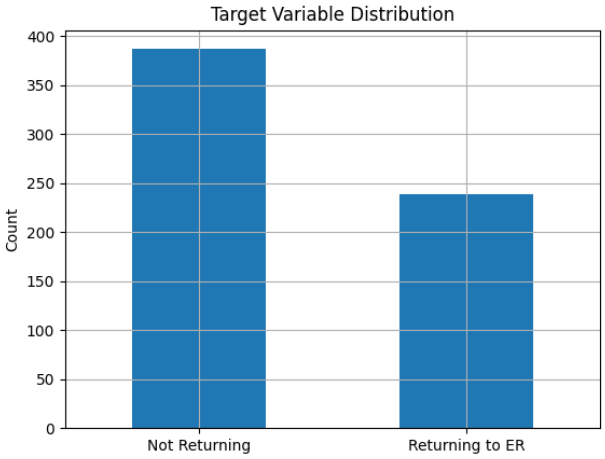

Figure 4.4: Target variable ER Sepsis

4.2.3 Preprocessing

For the purposes of our research, we convert the sequence of activities to static variables and treat the measurement variables as time series variables. Table 4.5 shows the sepsis dataset size before and after cases with missing values or cases with less than two Leucocytes time series variables (timesteps) were removed because as the change in Leucocytes cannot be measured with less than 2 measurements. It also shows the size of the training, test and validation sets.

Dataset	**Size**	Not Return ER	Return ER
Original dataset	1050		
After removing missing age and case_id	994		
After removing cases with no Leucocytes	956		
After removing sequences shorter than 2	626	387 (62%)	239 (38%)
Training set	415 (66%)	253 (61%)	162 (31%)
Test set	126 (20%)	82 (65%)	44 (35%)
Validation set	85 (13.5%)	52 (61%)	33 (39%)

Table 4.5: Dataset Sizes

For processing features, categorical features were converted to binary features. Age was normalized and the timestamp was converted to the number of seconds since admission.

To reduce the training time, we decided to include the most informative features. To achieve this we computed a correlation matrix using the Pearsons correlation coefficient. If two features are identical or highly similar, one of them can be removed as it adds little to no extra information [73]. By removing it, model complexity is reduced, as the model does not have to compute weights for it. In our experiments, if features had a higher absolute correlation of 0.7 with any other variable, it was excluded from the model training. Figure 4.5 shows the correlation coefficients between the different features.

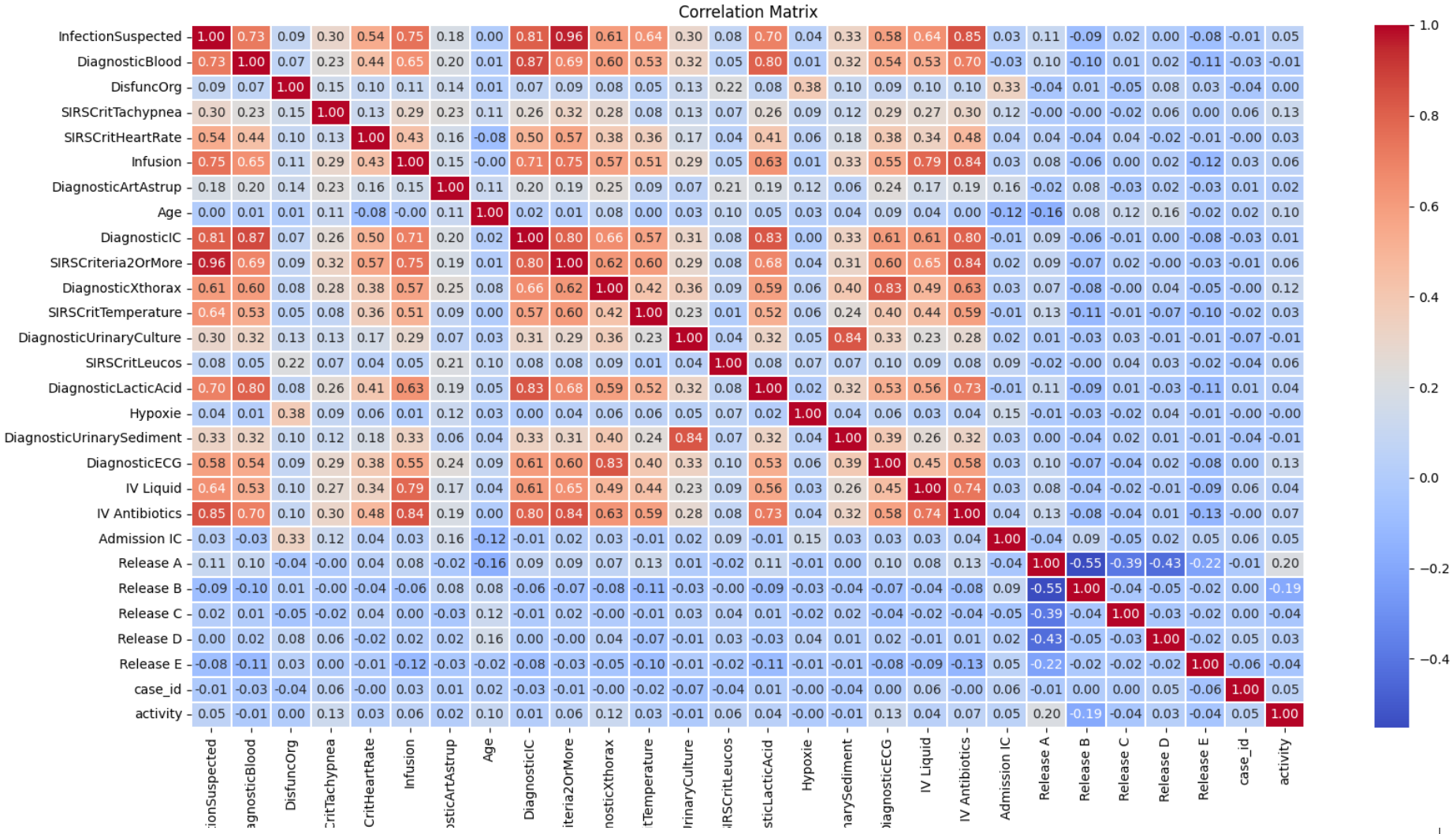

Figure 4.5: Correlation coefficient matrix for the sepsis data

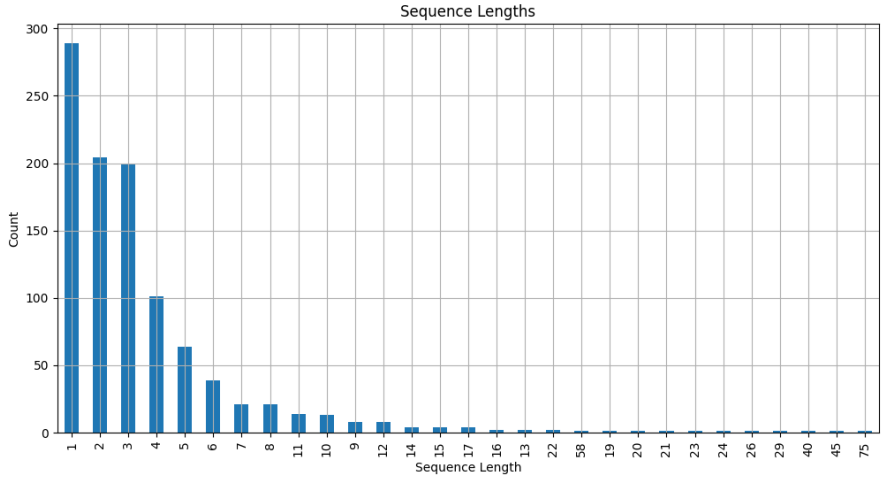

Figure 4.6: Sequence lengths Sepsis

It was decided to include all data of all cases which had a sequence larger than 1, as this allows to include the most cases. In the experiments, sequences were padded to have at least 5 timesteps. Figure 4.6 shows a number of cases with sequence lengths in the dataset.

4.2.4 Experiments

Since the Sepsis experiment is concerned with predicting whether or not a patient will return to the ER or not, it is a classification problem. As stated in Section 3.1, for classification problems the decoder RNN component outputs a single vector of shape $(d, 1)$ which is fed into a shallow NN, having a number of hidden units equal to the number of target classes, which in this case is 2. We will test the difference in performance between the attention, hybrid_1_step, hybrid_every_step, meta_1_step, meta_every_step architectures, explained in Section 3.3.

4.3 Fibonacci dataset

The previous cases showed that while we can apply the models to the data, the effectiveness of the attention mechanism is still unclear. In the case of the Sepsis data, we found that the effectiveness of deep learning models in general is limited. To specifically test the additional mechanism, we now

generate a dataset where there is a relationship between subsequent values in a sequence, making it a suitable problem for recurrent neural networks, as well as a relationship between the sequence and the static data, allowing to investigate the attention mechanism in a better context.

A dataset comprising of a variant of the Fibonacci sequence [2] was created, including static and time series data. The original Fibonacci sequence is a sequence of integers (elements) where each integer is the sum of the preceding two values. The variant we have created includes characteristics which describe the relationship between the input and output sequence, such as whether the sequence is in reverse, has been multiplied by a scalar, contains noise and the number of elements left out between the two sequences. A sample of the dataset can be seen in Appendix 8.5.1.

The objective of the dataset is to use the input Fibonacci sequence ($X_{sequence}$) as well as characteristics of the sequence (X_{static}) to predict the next 10 elements in the sequence ($Y_{sequence}$).

The static data in this problem is critical to accurately predict the next sequence, which makes it a suitable problem for testing whether the architectures we are using can indeed learn the correlation between the static and time series data.

4.3.1 Background and validity

This dataset was generated using a python script [3]. The process of data generation is described below. It does not reflect any real-world scenario and was used as a way to evaluate the attention mechanism in a better context and validate whether our models generalize to different domains and larger datasets.

1. **Create static dataset**: For each row, all the variables listed in Table 4.6 which fall within the static dataset must be initialized. The static variables are initialized in the following way:

[2] `https://en.wikipedia.org/wiki/Fibonacci_sequence`
[3] `https://github.com/hen3kr8/Fibonacci-variant-generator`

- **Fib 1**: Selects a uniform random sample from the range 1 and 49.
- **Fib 2**: Selects a uniform random sample from the range 1 and 49.
- **Gap**$_{XY}$: Selects a uniform random sample from the range 0 and 9.
- **Noise present**: Samples either 0 or 1 from non-uniform distribution with $P_0 = 0.9$ and $P_1 = 0.1$. Sequences in which noise is present will have Gaussian noise added to the sequence with mean = **noise mean** and standard deviation = to **noise standard deviation**
- **Noise**$_{mean}$: if noise is present, then the noise mean is 1.
- **Noise**$_{std}$: if noise is present, then the noise standard deviation is a uniform random sample within the range 0 and 4.
- **Reversed**: Selects a uniform random sample from the range 1 and 0.
- **Multiplier**: Selects a uniform random sample from the range 1 and 4.

2. **Create the timeseries sequences**.
   - **Fib 1** and **Fib 2** form the first 2 elements of the fibonacci sequence. The next 18 (double the sequence length of 10) + Gap$_{XY}$ number of elements are generated by following the formula $F_n = F_{n-1} + F_{n-2}$. This gives a sequence of 20 + Gap$_{XY}$ elements.
   - All elements of the sequence are multiplied by the **multiplier** variable.
   - If **reverse = 1**, the sequence is placed in reverse order, meaning the sequence is decreasing starting from the largest element.
   - the sequence is split into $X_{sequence}$, the first 10 elements in the sequence, and $Y_{sequence}$, which starts from the 10th + Gap$_{XY}$ element and ends at the 20th + Gap$_{XY}$ element.
   - If present, the noise is added to the sequences. For each element in the sequence, a noise value is sampled from the

Gaussian($\mathbf{Noise}_{mean}$, $\mathbf{Noise}_{std}$) distribution, which is added to the element.

4.3.2 Variables

Since the dataset was generated and recorded, it does not contain any missing values. It does contain noise which is indicated by the 'Noise present' variable.

Variable name	Description	Datatype	Range	Dataset
Fib_1	1st element in sequence	Discrete	[1, 49]	Static
Fib_2	2nd element in sequence	Discrete	[1, 49]	Static
Gap_XY	Number of elements between the last element of $X_{sequence}$ and 1st element of $Y_{sequence}$	Discrete	[0, 9]	Static
Noise present	Whether the sequence is noisy. 10% of the sequences contain noise	Binary	[0, 1]	Static
Noise mean	Noise is sampled from a normal distribution with mean = 1 and varying standard deviation.	Discrete	[0, 1]	Static
Noise std	Standard deviation used to sample the noise.	Discrete	[0, 4]	Static
Reversed	Whether the sequence is in decreasing or not. If reversed, Fib_1, Fib_2 are the last elements in $Y_{sequence}$. Else first elements in $X_{sequence}$. 50% of sequences are reversed.	Binary	[0, 1]	Static
Multiplier	Scalar with which sequence has been multiplied.	Discrete	[0, 4]	Static
$X_{sequence}$	Elements of Fibonacci sequence used as input. All sequences length 10.			Timeseries
$Y_{sequence}$	Elements of the Fibonacci sequence to predict. All sequences length 10.			Target

Table 4.6: Variables used in Fibonacci dataset

4.3.3 Preprocessing

As the dataset was generated, it required little preprocessing. It contained no missing values and all sequences were of length 10, eliminating the need for padding. Discrete variables were scaled using a Min-Max scaler as well as the elements of the X- and $Y_{sequences}$ A correlation matrix was not calculated as the features are uncorrelated by design, with the exception of the noise variables. Seeing as the number of features are small, they will all be included in the model training process.

Dataset	**Size**
Original dataset	1000
Training set	664 (66%)
Test set	200 (20%)
Validation set	136 (13.5%)

Table 4.7: Fibonacci Dataset Sizes

4.3.4 Experiments

The Fibonacci experiment is a forecasting problem tasked with predicting a sequence of elements. As mentioned in Section 3.1, in forecasting problems the decoder component outputs a vector for every timestep of the sequence, resulting in a shape of (d, t). This output is fed to a shallow NN with a single hidden unit, tasked with reducing the dimensionality to $(1, t)$ in order to predict the final sequence of elements.

5. Results

In this section, we present results from each of the Sepsis, Fibonacci, CF forecasting and CF improvement classification experiments. The chosen hyperparameters used for each of the 6 architectures of the final experiments are shown, of which performance plots of the specific hyperparameter search can be found in the relevant Appendix section. Each experiment for each architecture was run 15 times.

For classification experiments, Sepsis and CF improvement, the mean and standard deviation of metrics test accuracy, F1 score, Precision, Recall and number of training epochs recorded during the experiment run are shown. For classification, a higher F1 score is preferred. For forecasting experiments, the Test MSE and the number of training Epochs recorded during the experiment are shown. Regarding MSE, a lower value indicates a higher accuracy.

Primary analysis involved a non-parametric Kruskal-Wallis test for statistical significance between architectures, followed by a post hoc Dunn's test or parametric Welch's t-test. Each experiment concludes with the presentation of the attention weights found by the attention architecture. This is discussed in the following section.

5.1 Sepsis

In the Sepsis experiment the architectures were tasked with predicting whether or not a patient would be readmitted to the emergency room. It is a binary classification problem and performance between architectures is compared using F1 score, where a higher value indicates better performance. Table 5.1 shows the chosen hyperparameters based on the hyperparameter search. Figures 8.2, 8.3 and 8.4 in Appendix 8.4.1 show the results of each hyperparameter search. Table 5.3 presents the results of Welch's t-test with Bonferroni correction for comparing the F1 scores between each architecture,

indicating statistically significant differences where $p < .001$, suggesting the rejection of the null hypothesis that two distributions are equal. Table 5.4 shows the effect sizes measured by Cohen's d between architectures, with effect sizes of 0.2, 0.5, and 0.8 indicating small, medium, and large differences, respectively [74]. The distributions of the F1 score of the architectures are visualized in Figures 5.1 and 8.5.

Model	Number of layers	Number of units per layer	Initial learning rate
attention	5	32	10^{-4}
hybrid_1	7	128	10^{-5}
hybrid_every	6	128	10^{-5}
meta_no_seq	6	16	10^{-3}
meta_every	2	128	10^{-5}
meta_1	6	128	10^{-5}

Table 5.1: Hyperparameters used for final experiment Sepsis ER

5.1.1 Results

We noted that the architectures were very susceptible to falling into the local optima of only predicting the majority class. Therefore we deemed F1 score a more valuable metric, as it better reflects the architecture's ability to differentiate the two target variables. The meta_1_step architecture achieved the highest F1 score, and the hybrid_1 achieved the highest test accuracy. The attention architecture achieved the lowest F1 score.

Model	Test Accuracy (SD)	F1 (SD)	Precision (SD)	Recall (SD)	Epochs (SD)
attention	59.4 (4.8)	56.6 (3)	59.9 (6.2)	59.3 (4.8)	170 (23)
hybrid_1	63.1 (1.9)	61.13 (3.4)	62.3 (3.4)	63.1 (1.9)	153 (4)
hybrid_every	58.7 (3.6)	58.96 (3.6)	64.6 (2.9)	58.7 (3.6)	153 (3)
meta_no_seq	59.8 (4.8)	60.33 (4.5)	63.2 (3.7)	59.8 (4.8)	186 (20)
meta_every	59.3 (2.0)	60.19 (1.9)	63.8 (1.9)	59.3 (2.0)	155 (6)
meta_1	62.4 (2.6)	**62.6 (2.7)**	64.0 (2.3)	62.4 (2.6)	154 (6)

Table 5.2: Percentage Mean and standard deviation of Sepsis Return ER Results

5.1.2 Statistical significance

A Kruskal-Wallis test indicated that there was a significant difference in the median F1 scores across the 6 architectures (H(5) = 20.605, N = 15, $p < .001$).

This effect was small (E_R^2 = 0.087). Post-hoc comparisons using a parametric Welch's t-test with a Bonferroni correction for multiple tests and Cohen's d for effect size indicated that the mean F1 score of the attention architecture was significantly different to the hybrid_1 ($p = .001$, $d = 1.42$), meta_every ($p = .001$, $d = 1.43$) and meta_1 ($p < .001$, $d = 1.71$) architectures. The effect sizes were large, indicating inferior performance by the attention architecture. Effect sizes measured by Cohen's d of 0.2, 0.5, and 0.8 indicate small, medium, and large differences, respectively [74].

	attention	**hybrid_1**	**hybrid_every**	**meta_no_seq**	**meta_every**
hybrid_1	**.001**				
hybrid_every	.058	.102			
meta_no_seq	.013	.588	.369		
meta_every	**.001**	.369	.258	.919	
meta_1	**<.001**	.509	.025	.272	.091

Table 5.3: Results from Welch's parametric t-test for independent samples with Bonferroni correction ($\alpha_{adj} = 0.003$) on the F1 scores.

	attention	**hybrid_1**	**hybrid_every**	**meta_no_seq**	**meta_every**
hybrid_1	1.423				
hybrid_every	0.723	-0.617			
meta_no_seq	0.981	-0.200	0.334		
meta_every	1.433	-0.335	0.424	-0.038	
meta_1	1.710	0.244	0.864	0.41	0.644

Table 5.4: Results from Cohen's d measure for effect size based on the F1 scores. Absolute values 0.2, 0.5, and 0.8 indicate small, medium, and large differences, respectively [74].

5.1.3 Attention weights

Figures 5.2 and 5.3 show the attention weights of the model which had the highest F1 score during the experiments run after the hyperparameter search. Figure 5.2 shows the attention weights per timestep for all 126 test samples, whilst Figure 5.3 shows the average weights across all timesteps. Note that only the last 4 timesteps were used in the modelling process The weights are calculated by multiplying the learned representations from the RNN and NN models and applying a softmax activation function to the result (For more details, see Appendix 8.1). We can see from the figures that timestep 3, the last timestep, has the largest weight, followed by timesteps 2, 0 and 1.

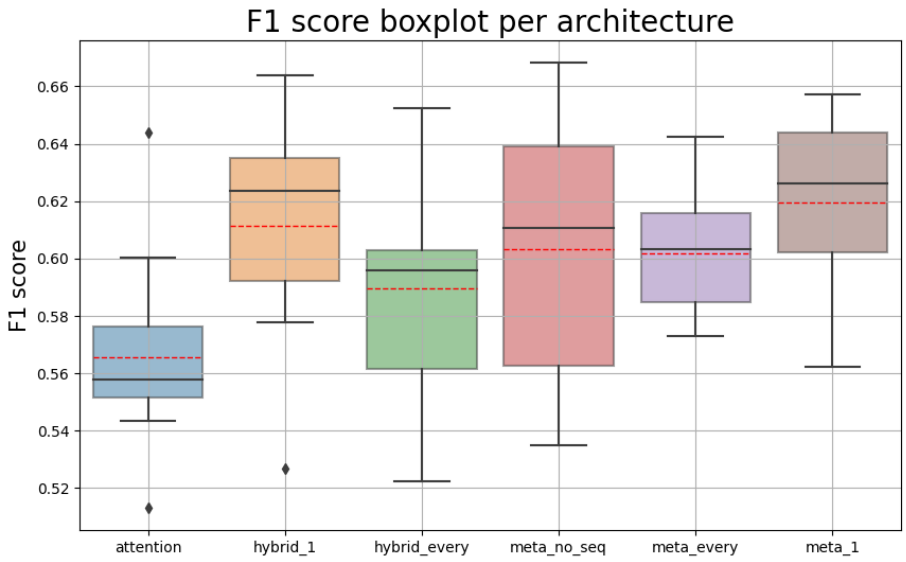

Figure 5.1: Boxplot for F1 score for Sepsis ER classification (Red line indicates mean).

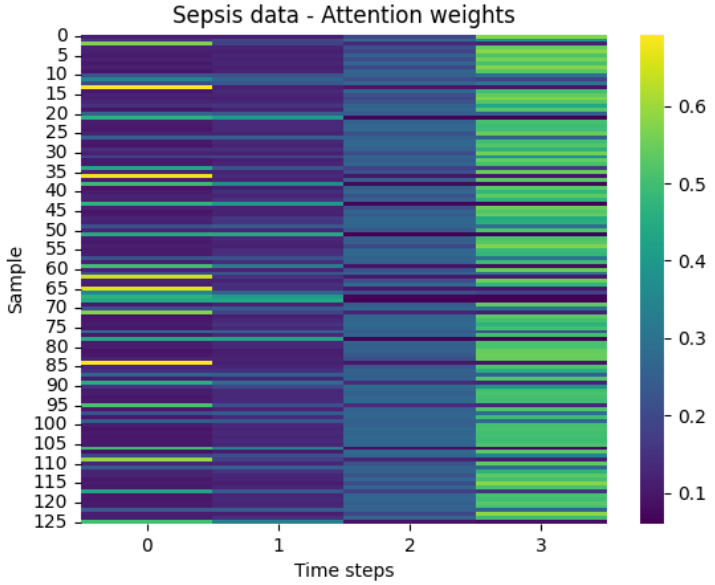

Figure 5.2: Attention weights for each test sample per timestep for the sepsis data

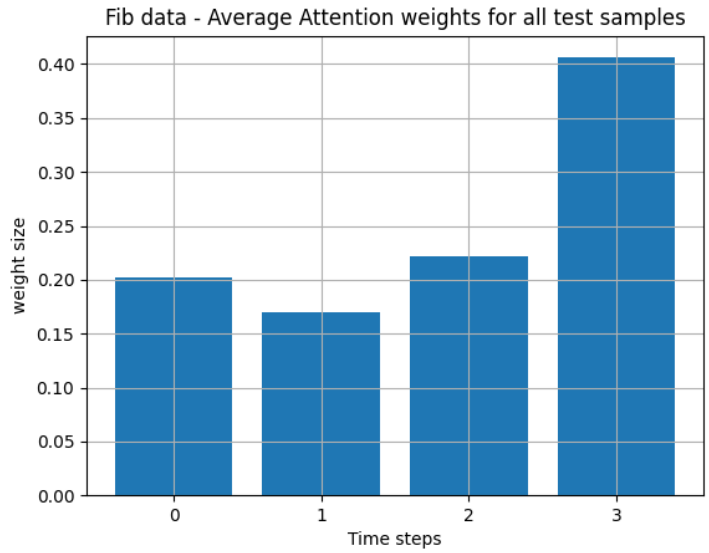

Figure 5.3: Average attention weights per timestep for the sepsis data

5.2 Fibonacci

In the Fibonacci experiment the architectures are tasked with forecasting the next 10 elements in the sequence. Its performance is evaluated by mean squared error, where a lower error indicates a better performance. Table 5.5 shows the selected hyperparameters based on the hyperparameter search. Figures 8.6, 8.7 and 8.8 in Appendix 8.5.2 show the plots of each individual hyperparameter search. The mean and standard deviation of the forecasting performance metrics, test MSE and number of training epochs of the final experiment, are shown in Table 5.6. The distributions of the test MSE of the architectures are visualized in Figures 5.4 and 8.9.

Model	Number of NN layers	Number of NN units	Initial learning rate
attention	2	64	10^{-4}
hybrid_1_step	2	64	10^{-4}
hybrid_every_step	2	32	10^{-4}
meta_no_seq	2	128	10^{-4}
meta_every_step	3	64	10^{-4}
meta_1st_step	2	128	10^{-4}

Table 5.5: Hyperparameters used for final experiment Fibonacci dataset

5.2.1 Results

Table 5.6 shows that the meta_1 architecture achieved the lowest test MSE, which means it had the lowest error among the models. Unsurprisingly the meta_no_seq architecture achieved the highest MSE by factor 100, making it the least accurate. It also took the largest number of epochs to train.

Model	Test MSE	Epochs
attention	3.69×10^{-5} (1.35×10^{-5})	224 (44)
hybrid_1	3.18×10^{-5} (9.46×10^{-6})	213 (52)
hybrid_every	9.21×10^{-5} (1.05×10^{-4})	216 (60)
meta_no_seq	1.59×10^{-3} (1.59×10^{-5})	294 (80)
meta_every	4.68×10^{-5} (1.70×10^{-5})	222 (75)
meta_1	3.12×10^{-5} (1.03×10^{-5})	207 (36)

Table 5.6: Mean and standard deviation of Fibonacci results.

5.2.2 Statistical significance

A Kruskal-Wallis test indicated that there was a significant difference in the median test MSE across the 6 architectures (H(5) = 46.079, N = 15, $p < .001$). This effect was small ($E_R^2 = 0.201$). Post-hoc comparisons using Dunn's test with a Bonferroni correction for multiple tests indicated that the mean test MSE was significantly different only for the meta_no_seq architecture.

	attention	hybrid_1	hybrid_every	meta_no_seq	meta_every
hybrid_1	.367				
hybrid_every	.297	.052			
meta_no_seq	**<.001**	**<.001**	**<.001**		
meta_every	.184	.025	.774	**<.001**	
meta_1	.345	.965	.047	**<.001**	.023

Table 5.7: p values according to Dunn's test ($\alpha_{adj} = 0.003$)

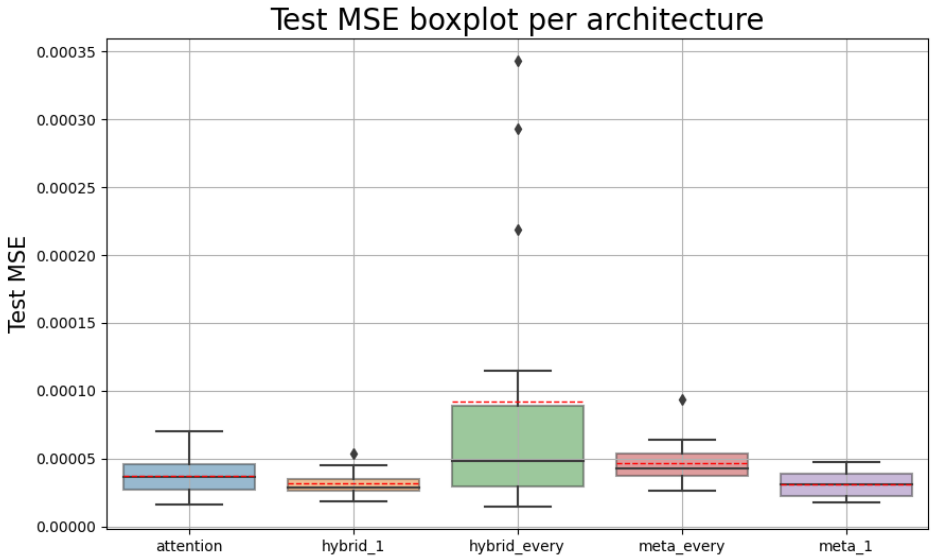

Figure 5.4: Boxplot for test MSE values for Fibonacci experiments, excluding meta_no_seq architecture (Red line indicates mean).

5.2.3 Attention weights

Figures 5.5 and 5.6 show the attention weights of the model which had the lowest MSE score during the experiments run after the hyperparameter search. Figure 5.5 shows the attention weights for each element for all 200 Fibonacci test sequences. Figure 5.6 shows the average attention weights for

each element of these test sequences. The highest weights were at elements 9, 8 and 7.

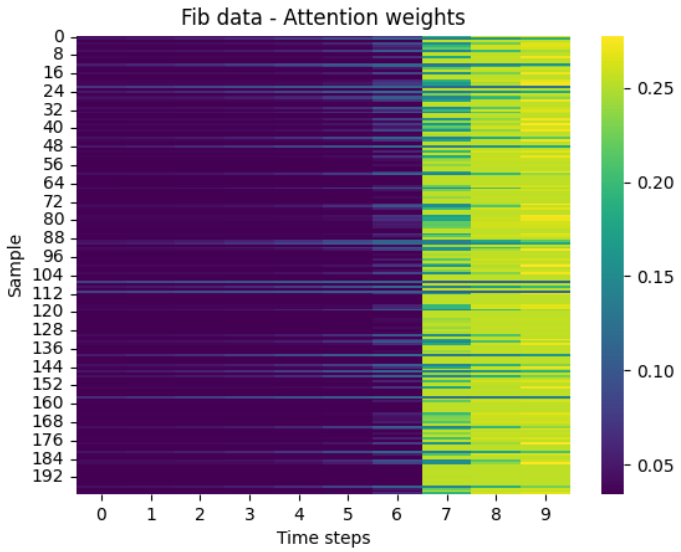

Figure 5.5: Attention weights for each test sample per timestep for the Fibonacci data

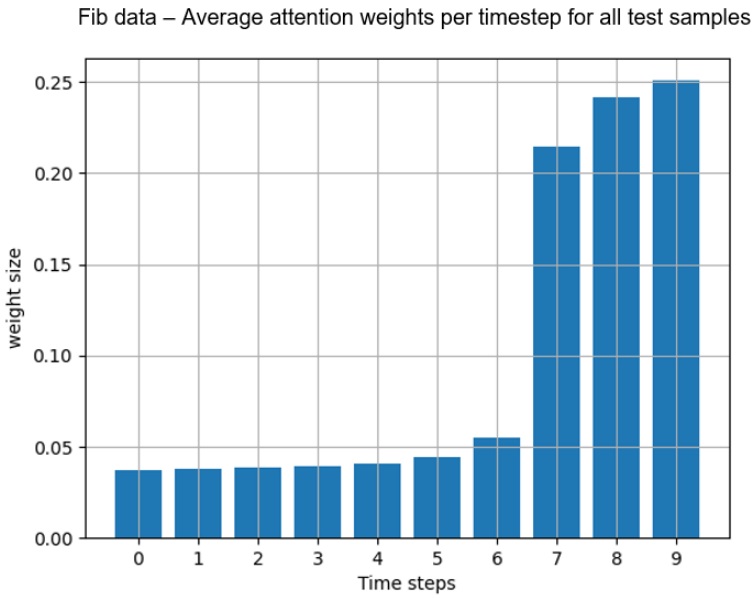

Figure 5.6: Average attention weights per timestep for the Fibonacci data

5.3 Cystic fibrosis forecasting

In the CF forecasting experiment the architectures are tasked with forecasting the next 5 ppFEV1 values of patients after they have taken Kaftrio. Its performance is evaluated by mean squared error, where a lower error indicates better performance. Table 5.8 shows the selected hyperparameters based on the hyperparameter search. Figures 8.10, 8.11 and 8.12 in Appendix 8.6.1 show the plots of each individual hyperparameter search. The mean and standard deviation of the forecasting metrics, test MSE and number of training epochs, of the final experiment are shown in Table 5.9. The distributions of the test MSE of the architectures are visualized in Figures 5.7 and 8.13.

Model	Number of layers	Number of units per layer	Initial learning rate
attention	6	32	10^{-6}
hybrid_1	2	32	10^{-4}
hybrid_every	2	16	10^{-4}
meta_no_seq	2	128	10^{-4}
meta_every	2	16	10^{-5}
meta_1	2	32	10^{-4}

Table 5.8: Hyperparameters used in the cystic fibrosis forecasting experiment

5.3.1 Results

Table 5.9 shows that the attention mechanism achieved the lowest MSE score, followed by the meta_every_step model. Both architectures had a high number of epochs in comparison to the other architectures, due to the smaller learning rate. Surprisingly, the meta_no_seq model achieved similar performance to the other architectures, despite not having a 2nd RNN component.

5.3.2 Statistical significance

A Kruskal-Wallis test indicated that there was a significant difference in the median MSE scores across the 6 architectures (H(5) = 63.281, N = 15, $p < .001$). This effect was small (E_R^2 = 0.278). Post-hoc comparisons using a non-parametric Dunn's test with a Bonferroni correction for multiple tests

Model	Test MSE	Epochs
attention	2.05×10^{-2} (1.17×10^{-3})	870 (61)
hybrid_1	2.61×10^{-2} (1.26×10^{-3})	574 (153)
hybrid_every	2.28×10^{-2} (1.19×10^{-3})	527 (196)
meta_no_seq	2.39×10^{-2} (1.01×10^{-3})	334 (114)
meta_every	2.13×10^{-2} (9.43×10^{-4})	853 (110)
meta_1	2.36×10^{-2} (1.30×10^{-3})	386 (101)

Table 5.9: Mean and standard deviation of CF forecasting results.

and Cliff's delta (δ) for effect size indicated that the mean MSE score of the attention architecture was significantly different to the hybrid_1 ($p < .001$, $\delta = 1.0$), hybrid_every ($p = .001$, $\delta = 0.84$), meta_1 ($p < .001$, $\delta = 0.947$) meta_no_seq ($p < .001$, $\delta = 0.84$) architectures. Effect sizes 0.15, 0.33 and 0.47 show small, medium and large differences [75], indicating that these effect sizes were large, and the attention architecture outperformed the other architectures.

	attention	hybrid_1	hybrid_every	meta_no_seq	meta_every
hybrid_1	**<.001**				
hybrid_every	**.001**	**<.001**			
meta_no_seq	**<.001**	.022	.110		
meta_every	.229	**<.001**	.056	**<.001**	
meta_1	**.001**	**<.001**	.371	.480	.005

Table 5.10: p values from Dunn's test (α_{adj} = 0.003)

Cliff's delta showed that the effect size measures between the attention and other architectures were large. Figures 5.7 and 8.13 emphasize the lower mean and median MSE of the attention architecture in comparison to other architectures.

	attention	hybrid_1	hybrid_every	meta_no_seq	meta_every
hybrid_1	1.0				
hybrid_every	0.840	-0.956			
meta_no_seq	0.947	-0.893	0.556		
meta_every	0.627	-1.0	-0.680	-0.876	
meta_1	0.840	-0.822	0.298	-0.262	0.796

Table 5.11: Effect size measures from Cliff's δ, which measures how often the values in one distribution are larger than values in a second distribution. Absolute effect sizes 0.15, 0.33 and 0.47 show small, medium and large differences [75].

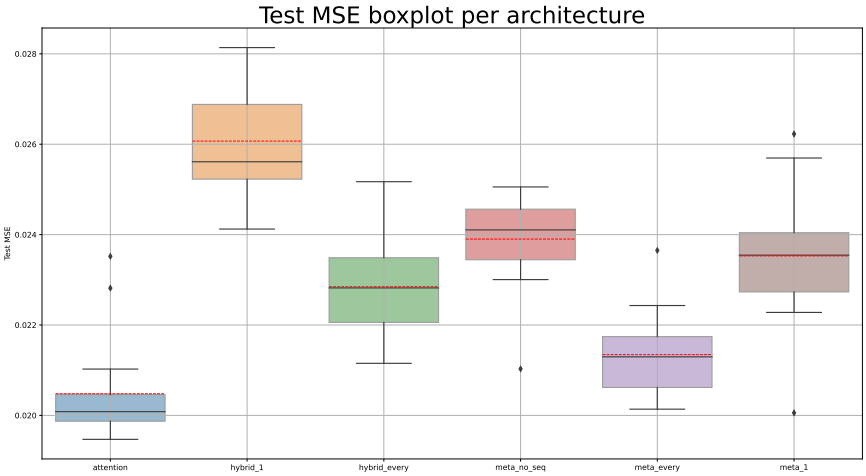

Figure 5.7: Boxplot for test MSE values for CF forecasting experiments (Red line indicates mean).

5.3.3 Attention weights

Figures 5.8 and 5.9 show the attention weights of the model which had the lowest MSE score in the experiments run after the hyperparameter search.

Figure 5.8 shows the attention weights for each element for all 33 CF forecasting test sequences. Figure 5.9 shows the average attention weights for each element of these test sequences. The highest weight was at timestep 0.

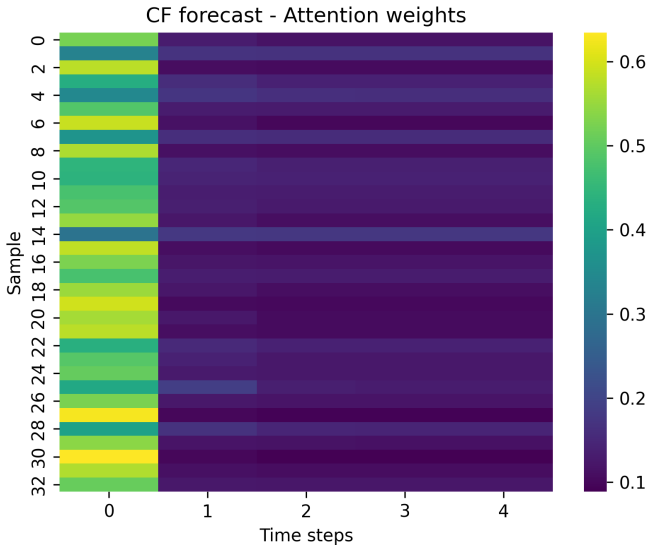

Figure 5.8: Attention weights for each test sample per timestep for CF forecasting data

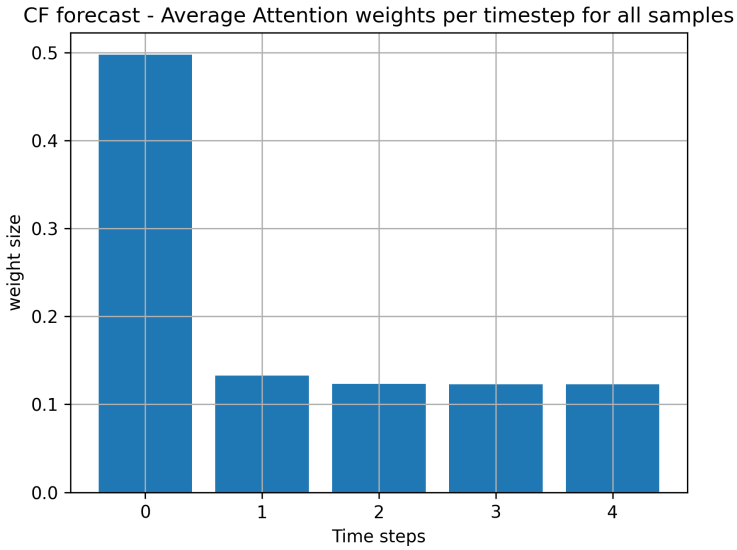

Figure 5.9: Average attention weights per timestep for the CF forecasting.

5.4 Cystic fibrosis improvement classification

In the CF improvement classification experiment, the architectures were tasked with predicting whether a patient's average ppFEV1 value after Kaftrio would be higher or lower than their last ppFEV1 value. It is a binary classification problem and performance between architectures is compared using F1 score, where a higher value indicates better performance. Table 5.12 shows the selected hyperparameters based on the hyperparameter search. Figures 8.14, 8.15 and 8.16 in Appendix 8.7.1 show the plots of each individual hyperparameter search. The mean and standard deviation of the classification metrics of the final experiment are shown in Table 5.13. The distributions of the F1 score of the architectures are visualized in Figures 5.10 and 8.17.

Model	Number of layers	Number of units per layer	Initial learning rate
attention	3	32	10^{-3}
hybrid_1	2	16	10^{-3}
hybrid_every	6	64	10^{-4}
meta_no_seq	6	32	10^{-3}
meta_every	7	128	10^{-4}
meta_1	4	32	10^{-4}

Table 5.12: Hyperparameters used in the cystic fibrosis improvement experiment.

5.4.1 Results

	Test Accuracy	F1	Precision	Recall	Epochs
attention	62.5 (9.0)	61.5 (5.9)	64.7 (7.1)	62.2 (9.1)	182 (16)
hybrid_1	67.6 (6.5)	67.8 (5.9)	69.6 (6.9)	67.6 (6.3)	172 (17)
hybrid_every	62.8 (6.4)	64.4 (5.8)	69.6 (3.8)	62.9 (6.3)	174 (31)
meta_no_seq	72.1 (6.7)	72.2 (6.2)	72.7 (6.3)	72.0 (6.4)	172 (19)
meta_every	62.8 (9.3)	64.4 (9.4)	69.4 (7.1)	62.7 (9.7)	167 (21)
meta_1	55.7 (8.8)	58.3 (8.6)	66.0 (5.6)	56.1 (9.2)	176 (23)

Table 5.13: Mean and standard deviation of CF improvement classification results

5.4.2 Statistical significance

A Kruskal-Wallis test indicated that there was a significant difference in the median MSE scores across the 6 architectures (H(5) = 27.539, N = 15,

$p < .001$). This effect was small ($E_R^2 = 0.118$).

Post hoc comparisons using a non-parametric Dunn's test with a Bonferroni correction for multiple tests and Cliff's delta (δ) for effect size indicated that the mean F1 score of the attention architecture was significantly different to the meta_no_seq ($p < .003$, $\delta = 0.8$) architectures. Effect sizes 0.15, 0.33 and 0.47 show small, medium and large differences [75], indicating that these effect sizes were large, and the attention architecture outperformed the other architectures.

	attention	**hybrid_1**	**hybrid_every**	**meta_no_seq**	**meta_every**
hybrid_1	.016				
hybrid_every	.240	.221			
meta_no_seq	**.001**	.144	.007		
meta_every	.122	.393	.711	.020	
meta_1	.439	**.002**	.051	**<.001**	.020

Table 5.14: Results from Dunn's test ($\alpha_{adj} = 0.003$)

	attention	**hybrid_1**	**hybrid_every**	**meta_no_seq**	**meta_every**
hybrid_1	0.551				
hybrid_every	0.320	-0.320			
meta_no_seq	0.800	0.396	0.631		
meta_every	0.356	-0.182	0.098	-0.542	
meta_1	-0.289	-0.653	-0.484	-0.796	-0.498

Table 5.15: Effect sizes according to Cliffs Delta.

5.4.3 Attention weights

Figures 5.12 and 5.11 show the attention weights of the model with the highest F1 score in the experiments run after the hyperparameter search. Figure 5.12 shows the attention weights for each timestep for all 34 test samples. The change in patient CFTR usage is plotted at the timestep it was started or stopped. Figure 5.11 shows the average attention weights for each element of these test samples. To the naked eye, there is no clear relationship between the CFTR usage and the attention weights. Overall, the weights seem to be decreasing in magnitude as the timesteps increase.

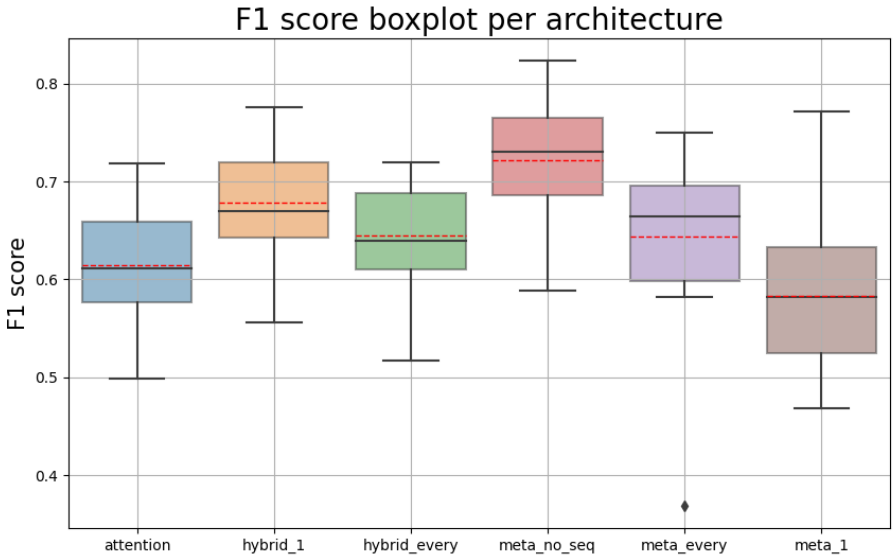

Figure 5.10: Boxplot for F1 score for CF improvement classification (Red line indicates mean).

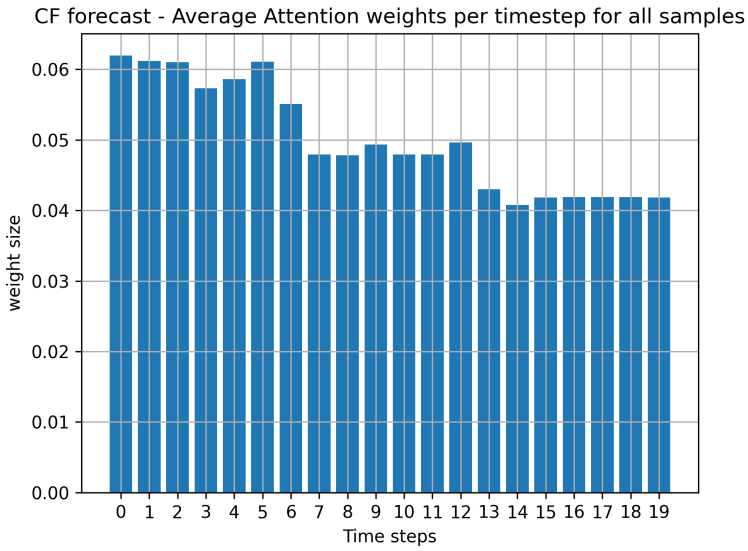

Figure 5.11: Average attention weights per timestep for the CF improvement classification.

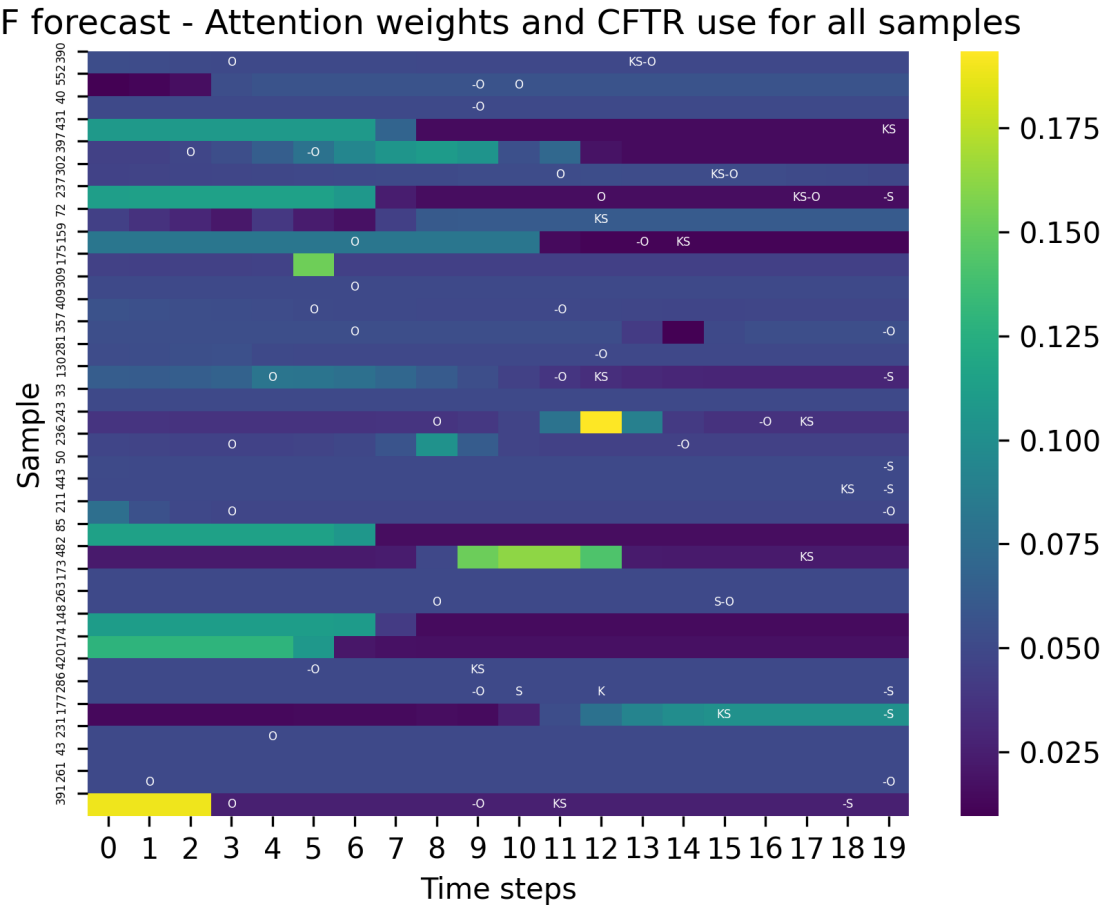

Figure 5.12: Attention weights per timestep for all samples with CFTR usage indicated. O = Orkambi start, -O = Orkambi stop. S = Symkevi start, -S = Symkevi stop. K = Kalydeco start, -K = Kalydeco stop.

6. Discussion

In this section of our work, we discuss the results from the specific experiments, the limitations of our work which impacted experiments, a general discussion in which we compare the different forecasting experiments to one another, as well as the classification experiments. The section concludes by discussing the advantages of the added attention mechanism to the architecture.

6.1 Experiment specific discussion

6.1.1 Sepsis

Evaluating Table 5.2, the test accuracy from the Sepsis experiment is better than random (59 - 63%). Architectures were susceptible to falling into the local optima of predicting the majority class, which achieves a 65% accuracy. This can be seen in the learning rate hyperparameter search (Figure 8.2) where the architectures trained using very large (10^{-3}) or very small (10^{-6}) initial learning rates all achieved a high test accuracy (65%) but a low F1 score (51%). The same can be said of the number of units hyperparameter search (Figure 8.4), where the architectures were more likely to fall into the majority class local optima if there were fewer hidden units per layer. This supported our usage of the F1 score rather than test accuracy to evaluate the architecture's ability to differentiate between the two classes.

The attention architecture achieved the lowest F1 score and was statistically significantly different to all other architectures with the exception of the hybrid_every and meta_no_seq model architectures (Table 5.3). The effect size measures between the attention architecture and all other architectures were large and positive (Table 5.4), indicating that the architectures that were significantly different outperformed the attention architecture by a considerable amount, and those that were not, likely would

have been significantly different had more samples been included.

The static data, which included variables such as whether a patient was admitted to the ICU, had a critical heart rate, or had a dysfunctional organ, was likely more informative than the time series data. This is based on the premise that the combination of these static variables likely provide greater insight into the likelihood of a patient returning to the ER compared to time series variables such as a patient's Leucocyte count. However, this assumption would benefit from further empirical validation to confirm the predictive superiority of static data over time series data in this context.

The attention mechanism effectively uses the cosine similarity to reduce the static data to a set of weights used for time series data, thereby placing much more emphasis on the time series data. This likely served as a disadvantage to the attention architecture, when the differential power lay in the static data. All other architectures having access to the learned representations of the static data in the decoder and final shallow NN achieved a higher F1 score, suggesting that that was the superior architecture setup at least for this dataset.

Although we cannot conclude that predicting the majority class is the global optimum of this problem as not all possible model hyperparameter configurations were explored, it is likely the case, seeing as no test accuracy higher than the majority class was achieved. Our experiment falls in line with the findings of [72], who, although they implemented a process model rather than deep learning models, could not find any hard rules to predict when a patient would return to the ER after 28 days.

Attention weights

As mentioned in Section 5.1.3, the last timestep, timestep 3, has the largest weight, followed by timesteps 2, 0, and 1. One obvious reason for this is that timesteps 2 and 3 are present in all samples, whilst timesteps 0 and 1 might be padding. This indicates the model has deemed the real timesteps of higher relevance than the padded data.

The time series value which was used in the modelling process was the

leucocyte variable. There was however no significant difference between the final leucocyte values of the different classes, meaning the large attention weight cannot be attributed to that. Given that the static data was more informative than the time series data, it is likely that the attention weights, which inform as to which time series element should be emphasized, are not very relevant.

6.1.2 Fibonacci dataset

The performance of all architectures on the test MSE was comparable, except for the meta_no_seq architecture, which deviated notably due to its distinctive design. Unlike the other architectures, meta_no_seq lacks a decoder RNN — a key component that significantly enhances a model's ability to predict sequences. This absence accounts for its poor performance, highlighting the decoder RNN's critical role in adding complexity and effectiveness to sequence-to-sequence prediction tasks, as established in the literature [76].

The test for significance suggests that there is a significant difference between at least one of the MSE distributions of the architectures, but the effect size suggests that this difference is small. Comparing the different architectures to one another using Dunn's test (Table 5.7) shows that there is no statistical difference between the MSE of the architectures, with the exception of meta_no_seq. This means the attention architecture achieved similar performance to the hybrid and meta architectures.

Attention weights

Referring to Figures 5.5 and 5.6, we observe that the final 3 elements have a higher weight than the rest. This is logical, as the first element of the $Y_{sequence}$ can be determined by adding, or subtracting, elements 8 and 9 in the $X_{sequence}$. This suggests that the attention mechanism has learned this inherent property of the dataset.

6.1.3 Cystic fibrosis forecasting

In this forecasting experiment the attention architecture scored the lowest MSE, and was significantly different with a large effect size in comparison to all other architectures. The meta_every architecture achieved the 2nd lowest MSE, and both architectures had a small learning rate, which is why it took more epochs to train. Smaller learning rates are less likely to overshoot the optima than larger learning rates, as well as more thoroughly exploring the search space.

The dataset for the CF forecasting experiment contains valuable static variables such as age, the number of pulmonary exacerbations (PEx), and sweat chloride concentration (SwCl) at baseline, which are associated with a patient's average ppFEV1 or ppFEV1 decline during Lumacaftor/Ivacaftor CFTR use [65]. The time series data contains the patient's CFTR usage over time and their initial ppFEV1 value, both of which influence a patient's ppFEV1 value over time. The attention mechanism effectively learns the relationship between static and time-series data, allowing it to predict the post-Kaftrio ppFEV1 over time with fewer errors compared to other architectures.

Attention weights

Figures 5.8 and 5.9 show that the 1st pre-Kaftrio measurement timestep had the highest weight. The 1st timestep is the 5th last timestep before a patient starts taking Kaftrio. The information in this timestep indicates which CFTRs a patient is currently using, their weight, whether they are using nebulized or oral anti-biotics and their ppFEV1 value. This information varies per patient.

A possible explanation for the high weight at this timestep is that the model only needs the 1st data point in the input sequence to forecast the output sequence. Since the pre-Kaftrio measurements only include the last 5 measurements, the patient's entire CFTR history is not fully captured.

While it might seem logical for the model to emphasize the final timestep before Kaftrio use, reflecting an imminent CFTR change, the attention

mechanism instead downplays its importance. This might be explained by the relatively stable nature of a patient's weight, suggesting that subsequent timesteps after the first may offer redundant information to the model.

6.1.4 Cystic fibrosis improvement classification

In the CF classification experiment, the meta_no_seq architecture achieved the highest F1 score and was significantly different to the attention and meta_1 architectures with a large effect. The meta_no_seq has only a single RNN component, which should make it less prone to overfitting and could explain its superior performance of 72%. This is however still less than predicting the majority class, which comprised 79% of the dataset.

The meta_no_seq's F1 score closely aligns with its test accuracy, suggesting balanced prediction capabilities across both classes without bias toward the majority class. Despite this, the relatively high variance in the results implies that incorporating more data could enhance prediction accuracy. The simplistic design of the meta_no_seq, having fewer model components and hence parameters than other tested architectures, raises the possibility that even simpler, possibly non-deep learning models might deliver superior performance. If the model were to surpass the baseline accuracy of predicting the majority class, its practical application should be considered.

Attention weights

The attention weights in Figures 5.12 and 5.11 seem to be decreasing over time. The weights per timestep vary in magnitude from 0.02 to 0.2. The exact weights per patient differ, but on average it seems the first 10 timesteps are more important than the last 10. The input time series data contains 20 timesteps compared to the 5 in the forecasting experiment (Figures 5.8 and 5.9) but the last 5 timesteps do not bear a resemblance to the weights in the forecasting experiment.

Since the input time series data contains more timesteps, it captures more of the patient's medical history, and importantly their historic CFTR usage.

Since the CFTR usage is more likely to change in the last 20 timesteps before a patient starts using the Kaftrio medicine compared to the last 5 timesteps, we expect the model to have a better chance of learning the importance of the CFTR change, and hence expect high attention weights at timesteps where a CFTR change occurred. At timestep 19, many patients stop the use of the Symkevi or Orkambi CFTR since they will start using Kaftrio at the following timestep. While we expected the attention weight here to be high, we see no clear relationship between CFTR usage and the attention weights, based on Figure 5.12. The attention weights seem to be the lowest at the final timestep, counter to our expectations. We notice that the weights do contain certain peaks (yellow in Figure 5.12), but they do not align with the CFTR change.

The architecture is trained on multiple variables (CFTR use, anti-biotic use, weight, and ppFEV1). The attention mechanism has likely determined that a combination of these variables rather than a specific timestep is of importance. This suggests that ppFEV1 does not solely depend on the CFTR change but perhaps on an interplay of multiple variables. This aligns with the multifactorial nature of ppFEV1 and reflects the complexity of the underlying health dynamics.

6.2 Limitations

This section outlines key limitations that influenced the scope and effectiveness of our work. These limitations range from the non-exhaustive search of hyperparameters to the constraints imposed by the batch size and sequence length in our models. Additionally, issues such as class imbalance in datasets may have affected the performance and outcomes of our experiments.

Hyperparameters The hyperparameter searches were not exhaustive. This limited the optimization of crucial parameters. These parameters include the number of units per layer, the dropout ratios, the count of RNN layers, and the selection of activation functions. By exploring a broader range and combination of these parameters in future studies, there is potential to

significantly enhance model performance and accuracy

Batch size Currently, our attention mechanism implementation does not support a batch size greater than 1. To make a fair comparison, all experiments were run using batch size 1. If the implementation were adjusted to support this, training time would decrease significantly, making it much more plausible to optimize the other parameters.

Sequence lengths The Fibonacci experiment suggested that including more extensive and longer sequences could potentially lower the MSE and improve model performance. In the cystic fibrosis forecasting experiment, the introduction of the Kaftrio medication in 2022[1], resulted in a limited number of post-treatment data points, with a significant difference in sequence lengths before and after treatment (See Figures 4.2 and 4.3). According to the architecture designs of the forecasting experiments, the input and output sequences must have the same length, as the decoder RNN is fed a concatenation of the output of the 1st RNN and NN component. The input and output sequences shorter than 5 were padded. Input sequences were padded with the value -1 and a masking layer ensured the architectures ignored these values. However, since the Tensorflow library does not support masking in the output layers, the output sequences were padded by repeating the final value in the sequence until the sequence was length 5. We realize this method of padding could impact the results and can introduce bias to the model to repeat predictions in the final stages of the forecast. A solution to this shortcoming is proposed as future work in Section 7.1.1.

Class imbalance A major problem of the sepsis experiment was that the models kept falling into local optima. The class imbalance of the data (65% vs 35%) likely attributed to this problem, and arguably the lack of data. Since samples in the sepsis data were only included if sequences were longer than 1, many samples were excluded. Including more samples of patients returning to the ER could improve performance. But as [72] also found, the

[1] https://www.umcutrecht.nl/nl/over-ons/nieuws/details/medicijn-kaftrio

data does not seem to be suited for predicting return to the ER, at least not using process mining or deep learning models.

The cystic fibrosis improvement classification dataset presented was even more imbalanced (79% vs 21%). Unlike the issues with local optima observed in the Sepsis datasets, the cystic fibrosis results displayed high variance. This variance stemmed from the small and imbalanced sizes of the test and validation sets. Increasing the size of these datasets is likely to enhance the model's accuracy, as a larger and more balanced dataset could provide a more representative sample of the population. This expansion would likely reduce the variance and improve the reliability of the results.

Note that the class imbalance impacted only the classification experiments. The Fibonacci and CF forecasting experiments did not have this limitation by design. This combination of experiments allowed for a more balanced overall view of the performance of the attention mechanism.

6.3 General discussion

Forecasting: CF vs Fibonacci

The mean test MSE across architectures in the Fibonacci experiment was much lower than in the CF forecasting experiment. Even the worst-performing meta_no_seq architecture in the Fibonacci experiment with its single RNN component achieved a lower test MSE (1.59×10^{-3}) than the best-performing attention architecture in the CF forecasting experiment (2.05×10^{-2}).

There are several reasons for the difference in performance:

1. The Fibonacci experiment had access to much more data (1000 data points vs 214). The sequences were also longer (10 vs 5).
2. It is a less complex problem, with data generated according to a known function, as opposed to the measured CF data which has many more variables at play, not all of which can be measured and is inherently less understood.

3. As with all real data, the measured CF data has challenges such as varying measurement frequencies, missing values, noise, and outliers. Although these aspects impact model performance, it more accurately mirrors the inconsistencies of the real world, and highlights the difference between real and generated data.

Classification: CF improvement vs Sepsis Return to ER

In the classification experiments, both datasets were imbalanced. The CF improvement dataset contained 79% positive and 21% negative cases, which made it more imbalanced than the Sepsis dataset's 62% negative and 38% positive cases. Yet the architectures in the CF improvement experiment were less susceptible to falling into local optima than in the Sepsis experiment based on the higher F1 scores in all architectures, with the exception of meta_1. The same holds for the test accuracy.

One reason for this difference in accuracy can be attributed to the relationship between the time series data and the target variable in each experiment. The relationship between previous ppFEV1 and CFTR usage to post-Kaftrio ppFEV1 is clear and supported by literature [66] (else the hospital would not distribute it). Whereas in the Sepsis experiment this evidence of whether or not a patient's Leucocytes influences their re-admittance to the ER is lacking.

Another difference lies in the variance of the results (Table 5.2 and 5.13). The CF improvement results have a higher standard deviation per architecture than the Sepsis results. This is very likely a cause of the small dataset used in the CF improvement experiment and suggests that including more data could improve accuracy.

1st step vs every step concatenation

In each of the experiments we also trained hybrid_1_step, hybrid_every_step, meta_1_step and meta_every_step architectures. In all of the experiments, we see that there is no significant difference in the performance between the meta_1_step and meta_every_step architectures. Regarding the hybrid

architecture, only in the CF forecasting experiment does hybrid_every_step significantly outperform the hybrid_1_step architecture.

This differs from [13, 20, 53] who found that concatenating the static data to the 1st step achieves better performance than concatenating it to every time step, but is likely due to the difference in the size of the data sets used. The mentioned works used sample sizes 3700, 8000 and 60,000 respectively compared to our 626, 1000, 214 and 224. This could suggest that concatenating the data to the first or every timestep has less influence on the model if the data set is small.

6.4 Advantage of an attention mechanism

We have seen that the attention architecture achieves a similar or even lower error regarding forecasting problems. We now explore possible reasons for this, as well as additional benefits of the attention mechanism.

The data dimensionality of the output of the attention mechanism is smaller than the output of the concatenation used in the meta and hybrid architectures. The output of the attention mechanism has shape (d, t), since it reduces the learned static representations to a set of attention weights to be multiplied with the time series representations. The meta-architecture (Figure 3.2) concatenates these learned representations (Figure 3.6), resulting in an output shape of $(2d, t)$. In the hybrid architecture (Figure 3.3) concatenation is performed between the learned static representations and the raw time series data, resulting in an output shape of $((X_t + d, t))$. The attention mechanism uses a smaller dimensionality, which leads to fewer model parameters to be learned in the RNN component, and the overall architecture (See a comparison of the learnable weights per architecture in Appendix 8.2). This reduction of dimensionality and model parameters decreases the likelihood of overfitting, without information loss. The meta_no_seq architecture is the exception to this as it has fewer parameters due to only a single RNN component. It does however have a higher dimensionality and achieves a higher error in comparison to attention architecture in sequence forecasting

experiments.

The attention mechanism provides a set of attention weights, which provides context for the model on which timesteps it should focus. This targeted focus allows the model to prioritize relevant data over less important information, thereby enhancing performance. Specifically, it helps in achieving lower error during training, allowing the model to predict more accurate sequences. In theory, the attention architecture should also converge faster, however, in none of the experiments did the attention architecture have even the lowest number of training epochs, even in experiments where it had a simpler hyperparameter setup (fewer layers, units per layer and larger learning rate) than the other architectures.

The attention weights can be visualised to provide potential insights about the structural dependencies of the data. In each experiment we have plotted the attention weights, which tell us which timesteps the architecture emphasizes when forecasting a sequence or making a classification prediction. We have plotted key events such as the CFTR change of CF patients to the attention weights, but saw no correlation. We did see in the Fibonacci experiment that the attention weights regard the final 2 timesteps as the most important, which suggests that the architecture has learned that it should utilize these two timesteps when predicting the output sequence. This reflects the underlying structure of the Fibonacci data, where adding the final 2 timesteps determines the following sequence element.

7. Conclusion

In this work, we examined whether integrating an attention mechanism into a model architecture that utilizes both static and time-series data results in improved accuracy and mean squared error. We compared this to similar ensemble and hybrid architectures without attention mechanisms across classification and forecasting tasks. Our experiments included two forecasting and two classification tasks. Regarding the forecasting experiments, the attention architecture achieved similar performance in the Fibonacci experiment and outperformed the architectures in the CF forecasting experiment. Conversely, in the sepsis and CF improvement classification tasks, meta and hybrid architectures proved to be superior.

Our findings suggest that the attention mechanism contributes to reduced mean squared error in forecasting problems compared to those without such mechanisms. That the attention architecture achieves better results than non-attention architectures with regards to forecasting falls in line with the work of [30] who found that adding an attention mechanism significantly increases performance in language translation. Therefore, we can conclude that, at least for forecasting problems, the attention architecture tends to achieve performance comparable to or better than hybrid and meta-forecasting methods when utilizing both static and time series data.

We have visualised the attention weights and found that for the CF classification experiment, they did not align with the known key events such as a change in CFTR. However, in the Fibonacci experiment these weights align with structural patterns of the dataset. Our findings suggest that the effectiveness of attention mechanisms in capturing structural patterns and key events depends on the complexity and context of the dataset. This does pave an interesting path to further research on whether attention mechanisms can be used to validate assumptions across diverse datasets.

7.1 Future Work

7.1.1 Varying sequences and the decoder RNN

The architectures presented in Section 3.3 consist of a NN, 2 RNNs and a shallow NN. The 2nd RNN takes as input the combined output of the NN and 1st RNN, combined either by attention or concatenation and outputs the output sequence. This requires the input and output sequences to be of the same length, which limits the flexibility of the architectures since the input sequence must be truncated/padded to the output length using the current Tensorflow and framework. Padding the output can introduce bias to the architecture, and should be minimized.

A good alternative to the current approach would be to add a decoder RNN model, a popular method in sequence-to-sequence modelling [76]. The decoder RNN would inherit the learned weights of the preceding components instead of their output predictions. Usually, the weights of an RNN are randomly initialized, but in this case, the initial weights are set equal to the weight of the preceding RNN. The decoder RNN then predicts the output sequence, starting with only the last timestep value in the input sequence. By decoupling the RNNs, the input sequence may have a different length than the output sequence. In the context of the Cystic fibrosis forecasting experiment, this modification would enable the inclusion of longer input sequences, capturing more of the patient's CFTR usage, and allowing the model to learn more correlations between CFTR change and ppFEV1 over time.

7.1.2 More data

Kaftrio treatment was approved for medical use in the EU in August 2020[1]. The WKZ hospital started dispensing Kaftrio to its patients in January 2022 as that was when the medication cost was included in the basic health insurance package. The available post-Kaftrio measurements used in the CF dataset

[1] https://www.ema.europa.eu/en/medicines/human/EPAR/kaftrio

were recorded from January 2022 to October 2023. It is therefore expected that the number of measurements is limited, as a patient typically visits the hospital once a quarter. The average number of post-Kaftrio measurements per patient was 5 (Figure 4.3). As the hospital continues treatment of Cystic Fibrosis, they will continue to record more measurements, resulting in longer output sequences for use in forecasting models. This additional data could provide deeper insights into the long-term impact of the CFTR modulator usage on lung function.

Other more data-rich domains can also be considered, such as weather data. Globally, weather data has been systematically collected for centuries, providing a wealth of historical data. It has a mix of static features, such as geographical location and elevation as well as time series features, temperature, humidity, and rainfall. This combination of static and time series data makes weather data well-suited to the architectures discussed in this work.

7.1.3 Focusing on the static data

In the Sepsis classification experiment, the static part of the dataset contained variables that were more informative than the Leucocytes the time series variable. Whilst the attention mechanism combined the learned static and time series representations to provide context to the time series data, it may be more beneficial for the Sepsis experiment to swap these roles around, and provide context for the static data, since this likely bears more informative power to the target variable.

Bibliography

[1] Gopinath Rebala, Ajay Ravi, and Sanjay Churiwala. In: *An Introduction to Machine Learning*. Cham: Springer International Publishing, 2019.

[2] Meindert Niemeijer et al. "Automated Detection and Differentiation of Drusen, Exudates, and Cotton-Wool Spots in Digital Color Fundus Photographs for Diabetic Retinopathy Diagnosis". In: *Investigative Ophthalmology & Visual Science* 48.5 (May 2007), pp. 2260–2267.

[3] Yann LeCun, Y. Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521 (May 2015), pp. 436–44.

[4] Amina Adadi and Mohammed Berrada. "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)". In: *IEEE Access* 6 (2018), pp. 52138–52160.

[5] Gabriëlle Ras, Marcel van Gerven, and Pim Haselager. "Explanation Methods in Deep Learning: Users, Values, Concerns and Challenges". In: *Explainable and Interpretable Models in Computer Vision and Machine Learning*. Springer International Publishing, 2018, pp. 19–36.

[6] David Martens and Foster Provost. "Pseudo-Social Network Targeting from Consumer Transaction Data". In: (Sept. 2011).

[7] M.J. Evans and J.S. Rosenthal. *Probability & Statistics - The Science of Uncertainty*. New York: W.H. Freeman and Company, 2004.

[8] Geoffrey I. Webb. "Overfitting". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 744–744.

[9] Haider Allamy. "methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)". In: (Apr. 2021).

[10] Lutz Prechelt. "Early Stopping — But When?" In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67.

[11] David Ruppert and Raymond Carroll. "Theory & Methods: Spatially-adaptive Penalties for Spline Fitting". In: *Australian & New Zealand Journal of Statistics* 42 (June 2000), pp. 205–223.

[12] Anna Leontjeva and Ilya Kuzovkin. "Combining Static and Dynamic Features for Multivariate Sequence Classification". In: Oct. 2016, pp. 21–30.

[13] Chen Lin et al. "Early Diagnosis and Prediction of Sepsis Shock by Combining Static and Dynamic Information Using Convolutional-LSTM". In: *2018 IEEE International Conference on Healthcare Informatics (ICHI)*. 2018, pp. 219–228.

[14] Sina Ardabili, Amir Mosavi, and Annamaria Varkonyi-Koczy. "Advances in Machine Learning Modeling Reviewing Hybrid and Ensemble Methods". In: Aug. 2019.

[15] Qinzhong Hou et al. “An adaptive hybrid model for short-term urban traffic flow prediction”. In: *Physica A: Statistical Mechanics and its Applications* 527 (Apr. 2019), p. 121065.
[16] Vijay Huddar et al. “Predicting Complications in Critical Care Using Heterogeneous Clinical Data”. In: *IEEE Access* 4 (Jan. 2016), pp. 1–1.
[17] Qiufeng Wang, Mirror Xu, and Amir Hussain. “Large-scale Ensemble Model for Customer Churn Prediction in Search Ads”. In: *Cognitive Computation* 11 (Apr. 2019).
[18] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.
[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
[20] Oriol Vinyals et al. “Show and tell: A neural image caption generator”. In: June 2015, pp. 3156–3164.
[21] Scott M. Lundberg et al. “Explainable machine learning predictions to help anesthesiologists prevent hypoxemia during surgery”. In: *bioRxiv* (2017).
[22] Alana de Santana Correia and Esther Luna Colombini. “Attention, please! A survey of Neural Attention Models in Deep Learning”. In: *CoRR* abs/2103.16775 (2021).
[23] Sneha Chaudhari et al. *An Attentive Survey of Attention Models*. 2021.
[24] Kelvin Xu et al. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. 2016.
[25] Jiasen Lu et al. “Hierarchical Co-Attention for Visual Question Answering”. In: (May 2016).
[26] Tan Zhi-Xuan et al. “A Multimodal LSTM for Predicting Listener Empathic Responses Over Time”. In: May 2019, pp. 1–4.
[27] Zheng Zhong, Wei Zhang, and et al. “Dual Attention Network for Chest X-ray Diagnosis”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.
[28] Ozan Oktay et al. “Attention U-Net: Learning Where to Look for the Pancreas”. In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. 2018.
[29] Ayoub Benali Amjoud and M. AMROUCH. “Automatic Generation of Chest X-ray Reports Using a Transformer-based Deep Learning Model”. In: *2021 Fifth International Conference On Intelligent Computing in Data Sciences (ICDS)* (2021), pp. 1–5.
[30] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016.
[31] William Chan et al. “Listen, attend and spell: A neural network for large vocabulary conversational speech recognition”. In: Mar. 2016, pp. 4960–4964.
[32] Jianlong Fu, Heliang Zheng, and Tao Mei. “Look Closer to See Better: Recurrent Attention Convolutional Neural Network for Fine-Grained Image Recognition”. In: July 2017, pp. 4476–4484.

[33] Karol Gregor et al. *DRAW: A Recurrent Neural Network For Image Generation*. 2015.
[34] Jiaolong Yang et al. *Neural Aggregation Network for Video Face Recognition*. 2017.
[35] Tom M Mitchell. *Machine learning*. 1997.
[36] Jerome H. Friedman. "Recent Advances in Predictive (Machine) Learning". In: *Journal of Classification* 23.2 (2006), pp. 175–197.
[37] Ravinder Ahuja et al. "Classification and Clustering Algorithms of Machine Learning with their Applications". In: *Nature-Inspired Computation in Data Mining and Machine Learning*. Ed. by Xin-She Yang and Xing-Shi He. Cham: Springer International Publishing, 2020, pp. 225–248.
[38] Doriana Marilena D'Addona. "Neural Network". In: *CIRP Encyclopedia of Production Engineering*. Ed. by Luc Laperrière and Gunther Reinhart. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 911–918.
[39] Wajdan Algihab et al. "Arabic Speech Recognition with Deep Learning: A Review". In: June 2019, pp. 15–31.
[40] Sakshi Virmani and Shilpa Gite. "Performance of convolutional neural network and recurrent neural network for anticipation of driver's conduct". In: July 2017, pp. 1–8.
[41] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.
[42] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.
[43] Gérard Biau, Luc Devroye, and Gäbor Lugosi. "Consistency of random forests and other averaging classifiers." In: *Journal of Machine Learning Research* 9.9 (2008).
[44] Dieu Bui et al. "Shallow Landslide Prediction Using a Novel Hybrid Functional Machine Learning Algorithm". In: *Remote Sensing* 11 (Apr. 2019), p. 931.
[45] Binh Pham et al. "Landslide Susceptibility Modeling Using Reduced Error Pruning Trees and Different Ensemble Techniques: Hybrid Machine Learning Approaches". In: *Catena* (Dec. 2018).
[46] Leo Breiman. "Bagging predictors". In: *Machine learning* 24 (1996), pp. 123–140.
[47] Jerome H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232.
[48] David H. Wolpert. "Stacked Generalization". In: *Neural Networks* 5.2 (1992), pp. 241–259.
[49] Peter Bühlmann. "Bagging, Boosting and Ensemble Methods". In: *Handbook of Computational Statistics: Concepts and Methods*. Ed. by James E. Gentle, Wolfgang Karl Härdle, and Yuichi Mori. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 985–1022.

[50] Harris Drucker, Robert Schapire, and Patrice Simard. "Boosting Performance in Neural Networks". In: *International Journal of Pattern Recognition and Artificial Intelligence* 07.04 (1993), pp. 705–719.

[51] Cristóbal Esteban et al. "Predicting Clinical Events by Combining Static and Dynamic Information Using Recurrent Neural Networks". In: *CoRR* abs/1602.02685 (2016).

[52] Wei Cao et al. "BRITS: Bidirectional Recurrent Imputation for Time Series". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018.

[53] Dingwen Li et al. "Integrating Static and Time-Series Data in Deep Recurrent Models for Oncology Early Warning Systems". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. Virtual Event, Queensland, Australia: Association for Computing Machinery, 2021, pp. 913–936.

[54] Feng Wang and David M. J. Tax. *Survey on the attention based RNN model and its applications in computer vision*. 2016.

[55] Haochao Ying et al. "Sequential Recommender System Based on Hierarchical Attention Network". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI'18. Stockholm, Sweden: AAAI Press, 2018, pp. 3926–3932.

[56] Caiming Xiong, Stephen Merity, and Richard Socher. "Dynamic Memory Networks for Visual and Textual Question Answering". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 2397–2406.

[57] Akira Fukui et al. *Multimodal Compact Bilinear Pooling for Visual Question Answering and Visual Grounding*. 2016.

[58] Xuelin Zhu et al. "Joint Visual-Textual Sentiment Analysis Based on Cross-Modality Attention Mechanism". In: *MultiMedia Modeling*. Ed. by Ioannis Kompatsiaris et al. Cham: Springer International Publishing, 2019, pp. 264–276.

[59] Hanbo Zhang et al. "Heterogeneous Feature Based Time Series Classification With Attention Mechanism". In: *IEEE Access* 11 (2023), pp. 19373–19384.

[60] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[61] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[62] Trista Hollweck. "Robert K. Yin. (2014). Case Study Research Design and Methods (5th ed.). Thousand Oaks, CA: Sage. 282 pages." In: *The Canadian Journal of Program Evaluation* 30 (Mar. 2016).

[63] Felix Mannhardt. *Sepsis Cases - Event Log*. en. 2016.

[64] J. S. Elborn. "Cystic fibrosis". In: *Lancet (London, England)* 388.10059 (2016), pp. 2519–2531.

[65] Danya Muilwijk et al. “Prediction of Real-World Long-Term Outcomes of People with CF Homozygous for the F508del Mutation Treated with CFTR Modulators”. In: *Journal of Personalized Medicine* 11 (Dec. 2021), p. 1376.

[66] Rebecca Voelker. “Patients With Cystic Fibrosis Have New Triple-Drug Combination”. In: *JAMA* 322.21 (2019), pp. 2068–2068.

[67] H. G. M. Heijerman et al. “Efficacy and safety of the elexacaftor plus tezacaftor plus ivacaftor combination regimen in people with cystic fibrosis homozygous for the F508del mutation: a double-blind, randomised, phase 3 trial”. In: *Lancet* 394.10212 (2019), pp. 1940–1948.

[68] Jonathan Guo, Anna Garratt, and Andrew Hill. “Worldwide rates of diagnosis and effective treatment for cystic fibrosis”. In: *Journal of Cystic Fibrosis* 21.3 (2022), pp. 456–462.

[69] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Hardcover published on 17 May 2013, Softcover published on 16 March 2019, eBook published on 17 May 2013. Springer New York, NY, 2013.

[70] H. P. Vinutha, B. Poornima, and B. M. Sagar. “Detection of Outliers Using Interquartile Range Technique from Intrusion Dataset”. In: *Information and Decision Sciences*. Ed. by Suresh Chandra Satapathy et al. Singapore: Springer Singapore, 2018, pp. 511–518.

[71] Mervyn Singer et al. “The Third International Consensus Definitions for Sepsis and Septic Shock (Sepsis-3)”. English. In: *JAMA* 315.8 (2016), pp. 801–810.

[72] F. Mannhardt and D. Blinde. “Analyzing the trajectories of patients with sepsis using process mining”. English. In: *RADAR+EMISA 2017, Essen, Germany, June 12-13, 2017*. CEUR Workshop Proceedings. RADAR + EMISA 2017 ; Conference date: 12-06-2017 Through 13-06-2017. CEUR-WS.org, 2017, pp. 72–80.

[73] Antnio Pacheco et al. *New Advances in Statistical Modeling and Applications*. Springer Publishing Company, Incorporated, 2014.

[74] Maciej Tomczak and Ewa Tomczak-Łukaszewska. “The need to report effect size estimates revisited. An overview of some recommended measures of effect size”. In: 21 (Jan. 2014), pp. 19–25.

[75] K. Meissel and E. S. Yao. “Using Cliff’s Delta as a Non-Parametric Effect Size Measure: An Accessible Web App and R Tutorial”. In: *Practical Assessment, Research, and Evaluation* 29.1 (2024), p. 2.

[76] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *ArXiv* abs/1409.3215 (2014).

[77] Ashish Vaswani et al. “Attention is all you need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010.

[78] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015.

[79] Graham L. Hall et al. “Official ERS technical standard: Global Lung Function Initiative reference values for static lung volumes in individuals of European ancestry”. In: *European Respiratory Journal* 57.3 (2021). Ed. by et al.

[80] Bernadette J. Prentice et al. “Cystic fibrosis-related diabetes and lung disease: an update”. In: *European Respiratory Review* 30.159 (2021).

8. Appendix A

8.1 Attention mechanism

Here follows the mathematics for calculating the attention weights, α, which are multiplied by the time series data before being fed into a final shallow neural network. The idea behind the attention mechanism is to map a query and a set of key-value pairs to an output [77], where the output is a weighted sum of values. This type of attention is known as scaled dot-product attention, as the dot product is used to measure the similarity between the query and the key to ultimately calculate the weights [77, 78].

Q : the final hidden layer from a neural network trained on static data
K, V : the final hidden state from recurrent neural network trained on time series data
w_Q, w_K, w_V : learnable weights updated during training for Q, K and V
n : the number of timesteps
d : the number of units in the hidden layer
$*$: element wise multiplication between matrices

$$Q = (1, d) = \begin{pmatrix} nn_1 & nn_2 & \cdots & nn_d \end{pmatrix}$$

$$K = V = (n, d) = \begin{pmatrix} t_1 rnn_1 & t_1 rnn_2 & \cdots & t_1 rnn_d \\ t_2 rnn_1 & t_2 rnn_2 & \cdots & t_2 rnn_d \\ \vdots & \vdots & \ddots & \vdots \\ t_n rnn_1 & t_n rnn_2 & \cdots & t_n rnn_d \end{pmatrix}$$

$$attention\,weights = softmax\left(\frac{QK^T}{\sqrt{d}}\right)$$

$$attention\,output = softmax\left(\frac{QK^T}{\sqrt{d}}\right) * V$$

$$
\begin{aligned}
softmax(QK^T) &= (1,d)(n,d)^T = (1,n) \\
&= softmax\left(\sum_{i=1}^{d} nn_i\, t_1 rnn_i \quad \sum_{i=1}^{d} nn_i\, t_2 rnn_i \quad ... \quad \sum_{i=1}^{d} nn_i\, t_n rnn_i\right) \\
&= \left(\alpha_1\, \alpha_2\, \cdots\, \alpha_n\right)
\end{aligned}
$$

$$
\begin{aligned}
\text{Repeat}(softmax(QK^T), d) &= (d,n) \\
\text{Repeat}(softmax(QK^T), d)^T * V &= (n,d) * (n,d) = (n,d) \\
\text{Sum over } d &:= (1,n) \\
&= \left(\alpha_1 \sum_{i=1}^{d} t_1 rnn_i \quad \alpha_2 \sum_{i=1}^{d} t_2 rnn_i \quad ... \quad \alpha_n \sum_{i=1}^{d} t_n rnn_i\right)
\end{aligned}
$$

8.2 Number of weights in the meta, hybrid and attention architectures

The meta, hybrid and attention architectures are similar but differ in how the static and time series data is concatenated and fed into the decoder RNN. In this section, we briefly discuss the different dimensionalities of the architectures before being fed into the decoder RNN. Whilst the meta_no_seq architecture has the least amount of learnable parameters, the attention architecture has the least amount regarding architectures that include a decoder RNN, which is more advantageous for forecasting. For large t, the number of parameters in the attention architecture is initially larger than the meta and hybrid, but as the number of hidden units increases, the number of weights in the meta and hybrid architecture increases more rapidly (See Figure 8.1).

X_s : The number of static features in the input data. The batch size is 1.
X_t : The number of time series features in the input data.
t : The number of timesteps per input sample.
l : The number of layers in the NN.
d : The number of hidden units per layer.
F_n : The number of hidden units in the shallow NN (1 layer).

Below we present how the number of weights for each component of each architecture was calculated. The neural network components are standard

Attention architecture	Input	Output	Number of weights
NN	$(X_s, 1)$	$(d, 1)$	$d\,[X_s + d(l-1) + l]$
*RNN	(X_t, t)	(d, t)	$d(X_t + d + 1)$
*Attention mechanism	$(d, t), (d, 1)$	(d, t)	$d(2t + 1)$
*Decoder RNN	(d, t)	$(d, 1)$	$d(X_t + d + 1)$
Shallow NN	$(d, 1)$	$(F_n, 1)$	$F_n(d + 1)$
Total number of weights *unique components			$2d^2 + 2dX_t + 3d + 2dt$

Table 8.1: Input, output shapes and number of weights per component for the attention architecture

Meta architecture	Input	Output	Number of weights
NN	$(X_s, 1)$	$(d, 1)$	$d\,[X_s + d(l-1) + l]$
*RNN	(X_t, t)	(d, t)	$d(X_t + d + 1)$
*Decoder RNN	$(2d, t)$	$(d, 1)$	$d(3d + 1)$
Shallow NN	$(d, 1)$	$(F_n, 1)$	$F_n(d + 1)$
Total number of weights *unique components			$4d^2 + 2d + dX_t$

Table 8.2: Input, output shapes and number of weights per component for the meta-architecture. Meta_no_seq only consists of the RNN component and has $d(X_t + d + 1)$ weights

Hybrid architecture	Input	Output	Number of weights
NN	$(X_s, 1)$	$(d, 1)$	$d\,[X_s + d(l-1) + l]$
*RNN	$(X_t + d, t)$	(d, t)	$d(X_t + 2d + 1)$
*Decoder RNN	(d, t)	$(d, 1)$	$2d^2 + d$
Shallow NN	$(d, 1)$	$(F_n, 1)$	$F_n(d + 1)$
Total number of weights *unique components			$4d^2 + 2d + dX_t$

Table 8.3: Input, output shapes and number of weights per component for the hybrid architecture

across all architectures, given that the number of layers is the same. Each term below is the sum of the weights between the input layers and the 1st hidden layer (1st parenthesis), the weights in the recurrent layer (RNN) or between the 1st and l layers (NN) (2nd parenthesis), and the bias.

$$
\begin{aligned}
NN : (X_s \times d) + [(d \times d) \times (l-1)] + (d \times l) &= d\,[X_s + d(l-1) + l] \\
RNN_{att} = RNN_{meta} : (X_t \times d) + (d \times d) + d &= d(X_t + d + 1) \\
RNN_{hyb} : [(X_t + d) \times d] + (d \times d) + d &= d(X_t + 2d + 1) \\
Att\_mechanism : (d \times t) + (d \times t) + (d \times 1) &= d(2t + 1) \\
Decoder\_RNN_{meta} : (2d \times d) + (d \times d) + d &= d(3d + 1) \\
Decoder\_RNN_{att} = RNN_{att} & \\
Decoder\_RNN_{hyb} : (d \times d) + (d \times d) + d &= 2d^2 + d \\
Shallow\_NN : (d \times F_n) + F_n &= F_n(d + 1)
\end{aligned}
$$

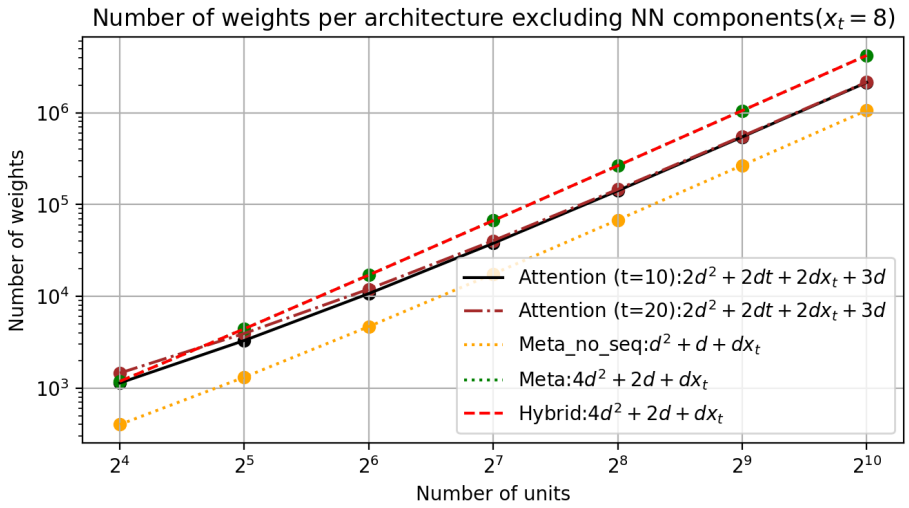

Figure 8.1: Number of weights per architecture as the number of hidden units increases. The NN components have been excluded as they have the same amount of weights across architectures

8.3 CF data: Further variable description and alignment

This section follows from Section 4.1 and serves as a supplementary text.

Below we show a synthesized sample of the static and time series data used in the experiment:

PATNUM	Weight	Height	Chloride-ZW	nebulized_-IV	oral_IV	diabetes	number_-admis-sions	Gender	Birth year	ppFEV1	has_-F508del
1	52	159	81	0	1	1	3	M	1996	59,4	1

Table 8.4: A sample of the static dataset used before scaling. This sample has been synthesized as to not show any private patient data.

PATNUM	DATUM	FEV1	ppFEV1	Kaftrio	Kalydeco	Symkevi	Orkambi	diabetes	nebulized_IV	oral_IV	Weight
11	2018-05-03	4,685	94,3	0	0	0	0	1	0	0	80
11	2018-06-29	4,703	95,9	0	1	0	0	1	0	0	81

Table 8.5: A sample of two successive measurements of a patient, before scaling is applied. This sample has been synthesized so as to not show any private patient data.

The preprocessing of the time series data differs from the static data in that each variable changes throughout a patient's CFTR modulator usage. This section describes how each variable's measurement date was aligned with the lungfunction (FEV1) measurement dates. Static variables are not described as they are date-independent.

For each table, a unique approach was selected to align its dates with the lungfunction measurement dates.

ppFEV1

The target variable, "percent predicted Forced Expiratory Volume in 1 second" is not recorded in the dataset but can be calculated as follows:

$$\text{ppFEV1} = \left(\frac{\text{Actual FEV1}}{\text{Predicted FEV1}} \right) \times 100 \tag{8.1}$$

The patient's actual FEV1 is recorded, and their Predicted FEV1, that is the FEV1 predicted for a healthy person of the same age, height, sex and ethnicity as the patient, can be calculated using reference equations [79] as specified by the Global Lung Function Initiative.

$$\text{Predicted FEV1} = exp\left(\alpha + (\beta \times \log(\text{height})) + (\gamma \times \log(\text{age})) + \delta + \epsilon\right) \tag{8.2}$$

Where the constants are dependent on the ethnicity, age, height and gender of a patient.

Apotheek table - CFTR usage

The *apotheek* table contains the CFTR modulator type, and the stop and start dates of patients. The CFTR modulator types present in the data are Kafrtio, Kalydeco, Symkevi, and Orkambi. To indicate exactly when a patient is using which modulator, this column is converted into 4 binary variables. This dataset is joined to the lung function measurement dataset on patient number and date. As the dates are rarely the same, the tables are aligned by joining each row in the *apotheek* dataset to the closest lung function date where the CFTR modulator date is less than the lung function date.

Diagnose - diabetes

Diabetes is a chronic metabolic disorder characterized by high blood sugar levels due to the body's inability to produce enough insulin or effectively use the insulin it produces. Cystic Fibrosis-related diabetes (CFRD) occurs in about 30% of patients and is associated with decreased lung function and an overall increase in mortality from lung disease [80]. Its presence in the static dataset indicates whether a patient has been diagnosed with diabetes before their first usage of CFTR modulators.

Medication - Nebulised and Oral antibiotics

These binary columns indicate the dates on which a patient was using antibiotics. Since antibiotic use is temporary, unlike diabetes, it is only marked with a 1 on the dates that a patient uses it. The dates of antibiotic use do not align with the lung function date. They are joined to the closest lung function date where the antibiotic usage date is less than the lung function date and still falls within a 30-day window. A window of 30 days was decided after consulting with a domain expert as the effect of the anti-biotics may significantly influence the patient's condition for up to 30 days.

Metingen - Height and Weight

Height and weight measurements were integrated with the lung function data to create a comprehensive dataset. In cases where the height or weight measurements were missing, the average of the patient's preceding and succeeding height and weight measurements was used, given that the missing measurement's date still falls within 180 days of the preceding and succeeding value. A window of 180 days was decided as in general a patient's height and weight do not change significantly during this time. Since height seldom changed between a patient's measurements, it was not included in the time series dataset.

8.4 Sepsis ER Experiment

8.4.1 Results from hyperparameter search

8.4.2 Results Sepsis ER classification per architecture

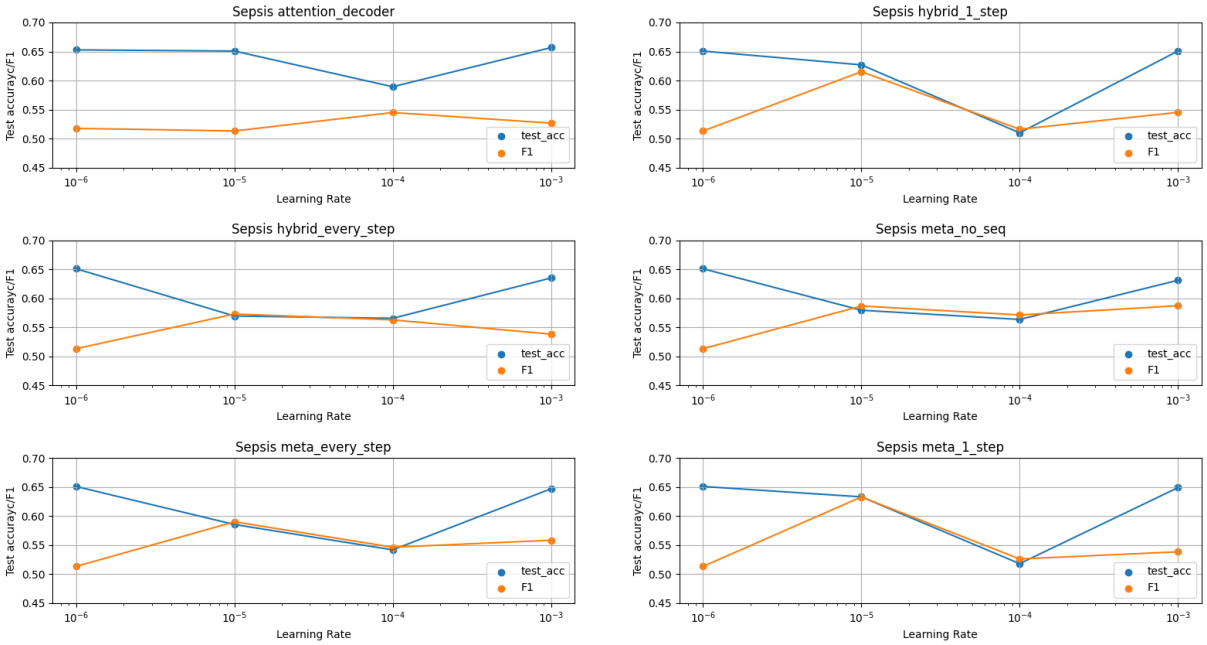

Figure 8.2: Test and F1 score per model per learning rate.

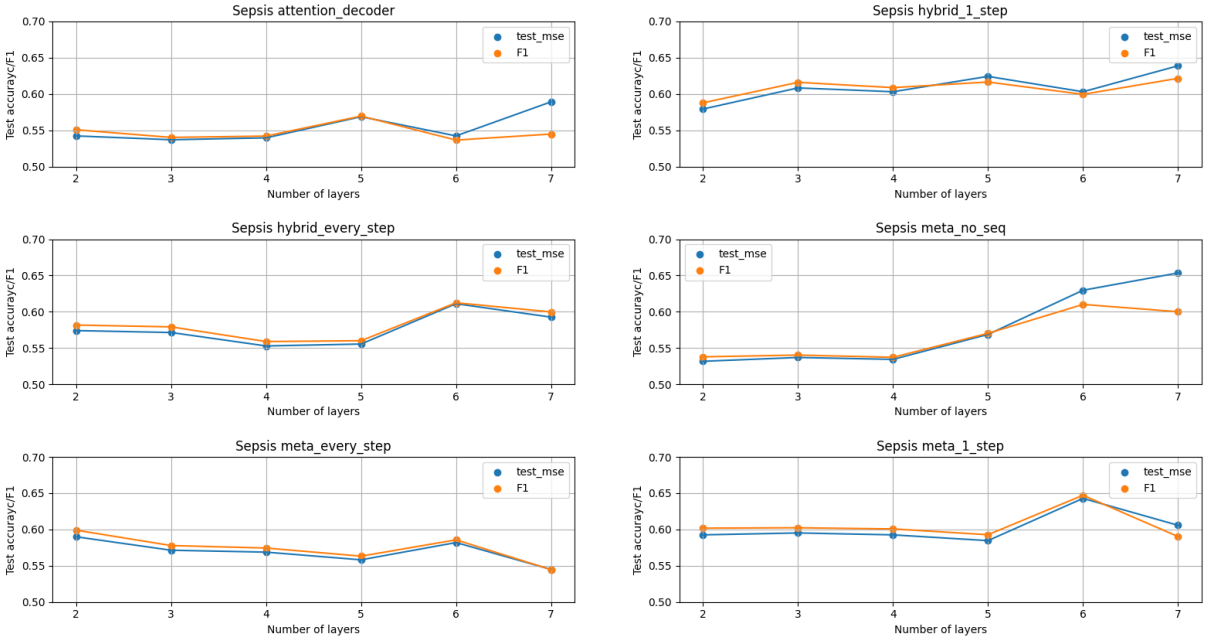

Figure 8.3: Test and F1 score per model per layer

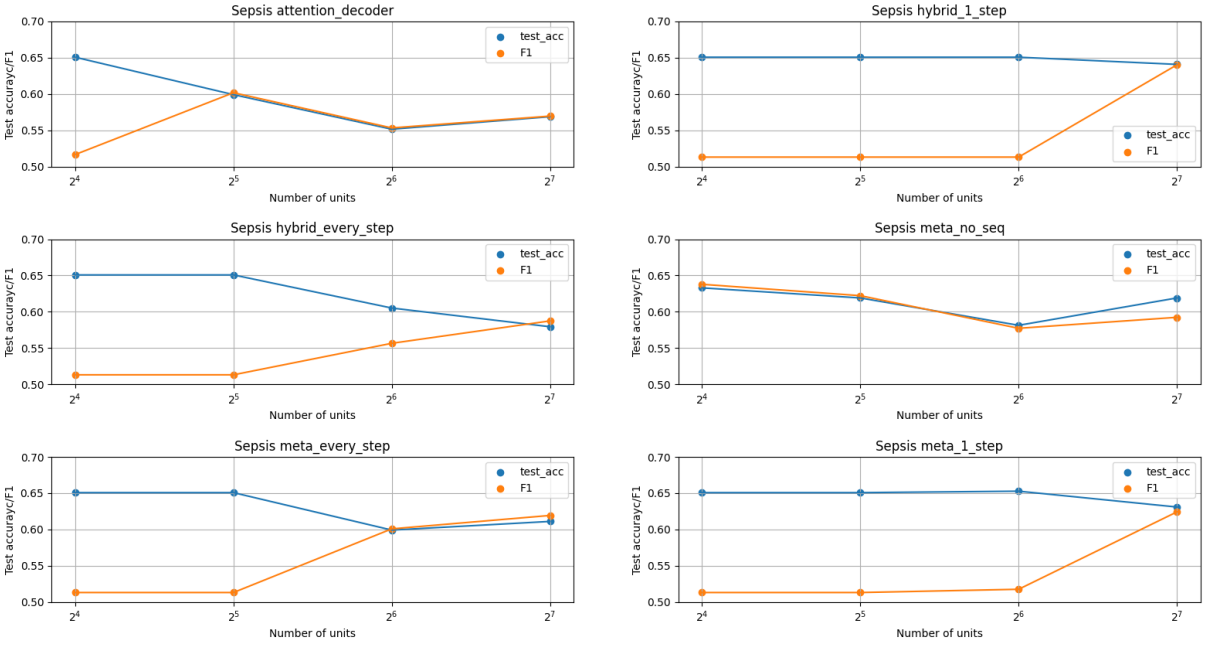

Figure 8.4: Test and F1 score per model per number of hidden units

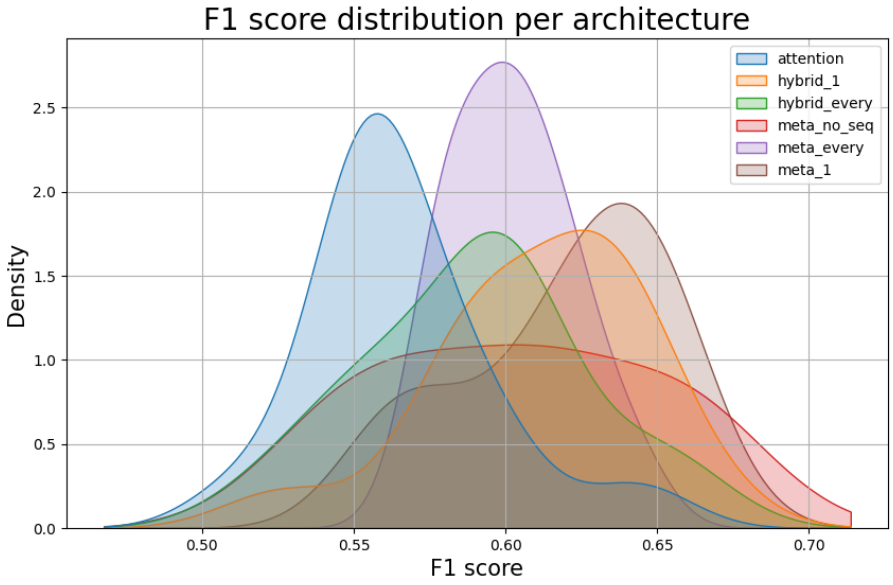

Figure 8.5: Density plots for F1 score for Sepsis ER classification.

8.5 Fibonacci experiment

8.5.1 Fibonacci datasample

Tables 8.6, 8.7, 8.8 show the static and time series data of the Fibonacci sequence variant used in training our models.

id	Fib_1	Fib_2	gap_XY	noise_present	noise_mean	noise_std	reversed	multiplier
0	24	18	0	0	0	0	0	1
1	31	39	3	0	0	0	1	4
2	19	21	6	0	0	0	1	2
3	17	15	5	0	0	0	0	4
4	3	43	6	1	1	4	1	1
5	37	9	1	0	0	0	0	1

Table 8.6: Static data in the variant Fibonacci sequence

id	Fib_1	Fib_2	Fib_3	Fib_4	Fib_5	Fib_6	Fib_7	Fib_8	Fib_9	Fib_10
0	24	18	42	60	102	162	264	426	690	1116
1	4120220	2546436	1573784	972652	601132	371520	229612	141908	87704	54204
2	4913034	3036422	1876612	1159810	716802	443008	273794	169214	104580	64634
3	68	60	128	188	316	504	820	1324	2144	3468
4	3365181	2079798	1285379	794413	490967	303445	187534	115912	71634	44273
5	37	9	46	55	101	156	257	413	670	1083

Table 8.7: Timeseries input ($X_{sequences}$) in the variant Fibonacci sequence

id	Fib_1	Fib_2	Fib_3	Fib_4	Fib_5	Fib_6	Fib_7	Fib_8	Fib_9	Fib_10
0	1806	2922	4728	7650	12378	20028	32406	52434	84840	137274
1	7908	4888	3020	1868	1152	716	436	280	156	124
2	2226	1376	850	526	324	202	122	80	42	38
3	62236	100700	162936	263636	426572	690208	1116780	1806988	2923768	4730756
4	1528	951	581	355	228	134	90	48	43	8
5	2836	4589	7425	12014	19439	31453	50892	82345	133237	215582

Table 8.8: Timeseries output ($Y_{sequences}$) in the variant Fibonacci sequence

8.5.2 Results from hyperparameter search

8.5.3 Results Fibonacci sequence forecasting per architecture

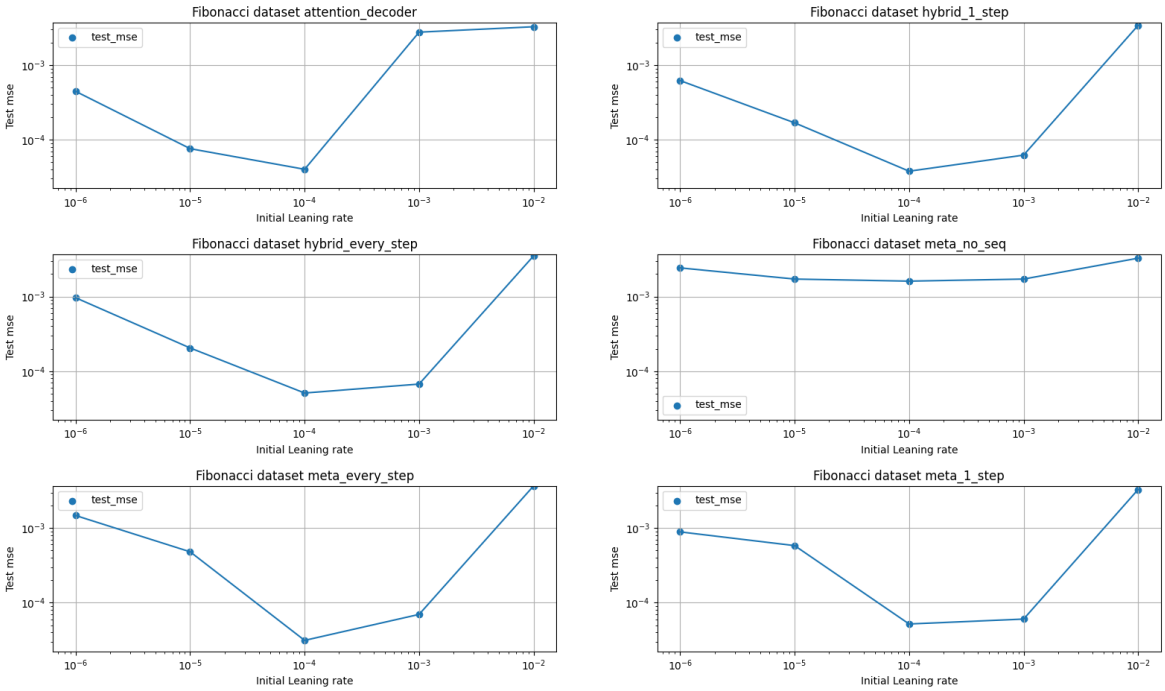

Figure 8.6: Test MSE per initial learning rate

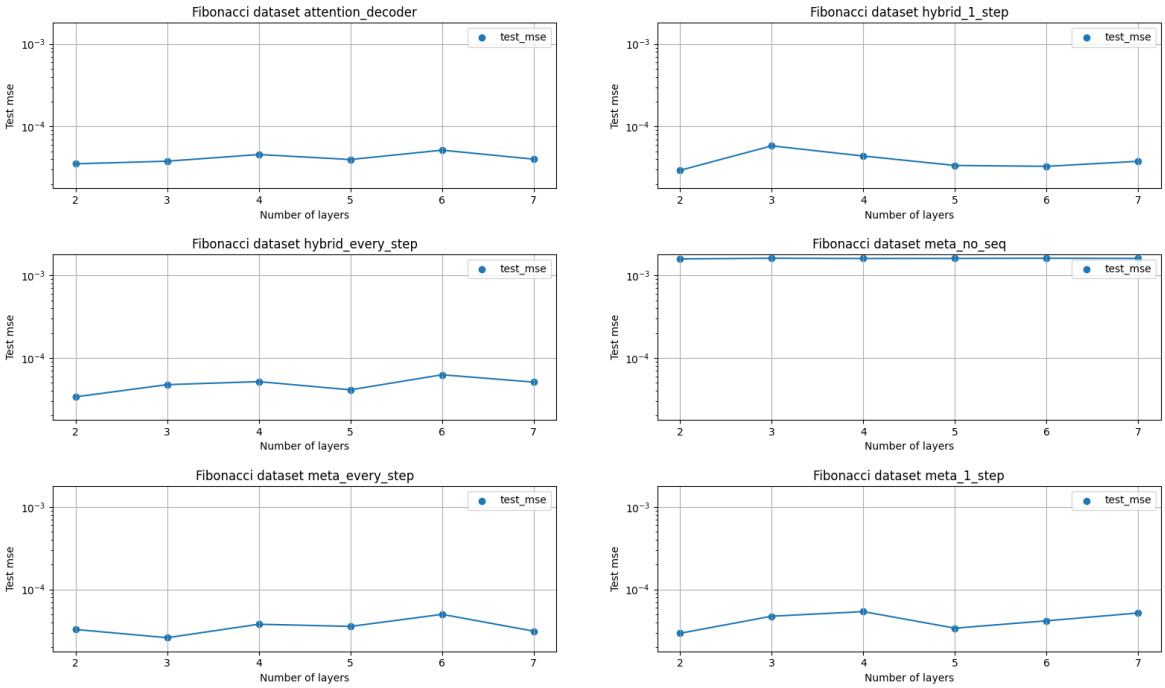

Figure 8.7: Test MSE per number of layers

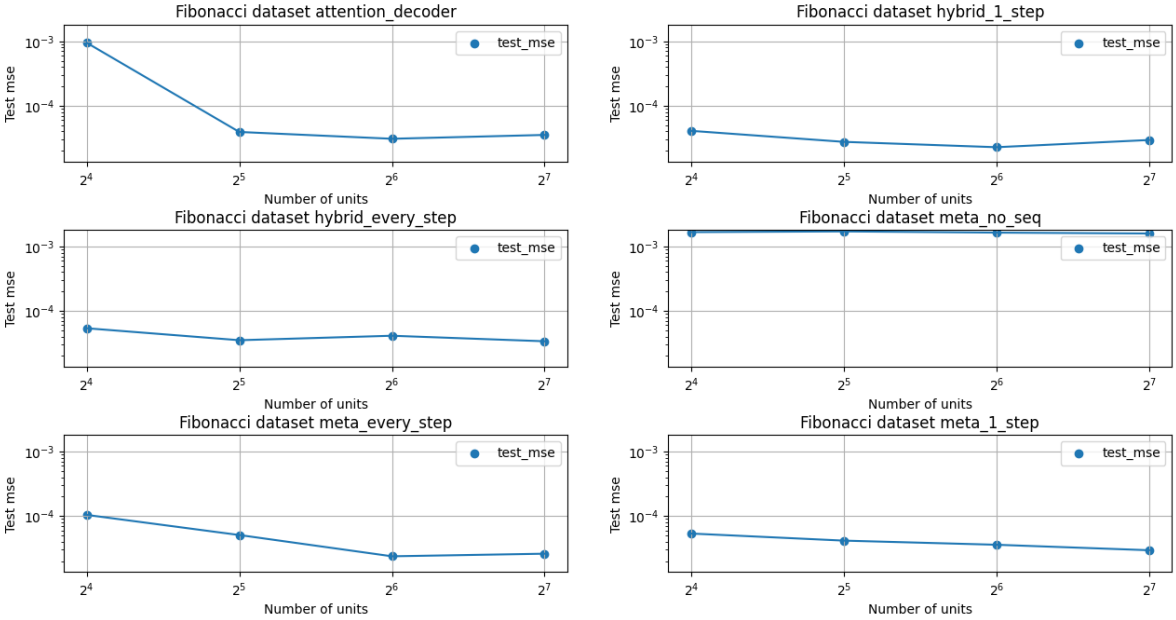

Figure 8.8: Test MSE per number of units

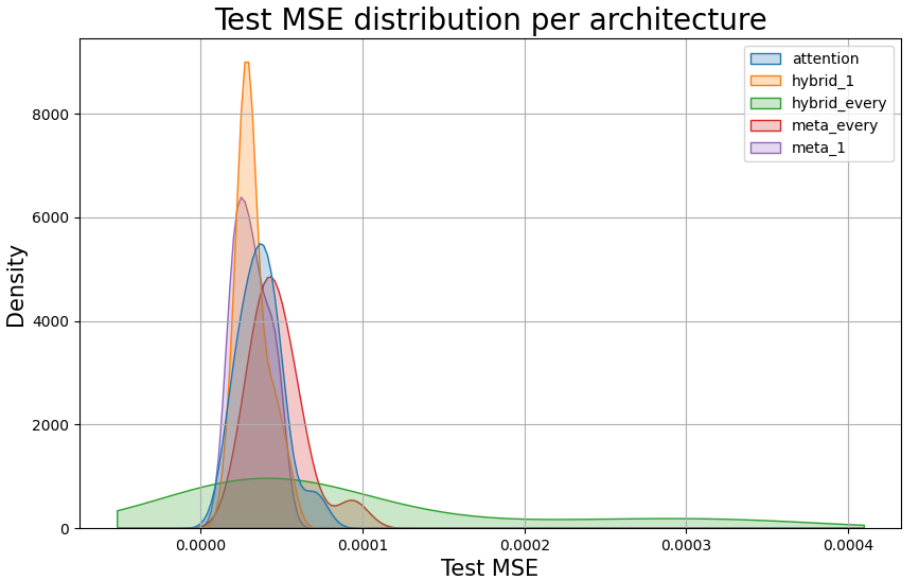

Figure 8.9: Density plots for test MSE values for Fibonacci experiments, excluding meta_no_seq architecture.

8.6 Cystic fibrosis forecast experiment

8.6.1 Results from hyperparameter search

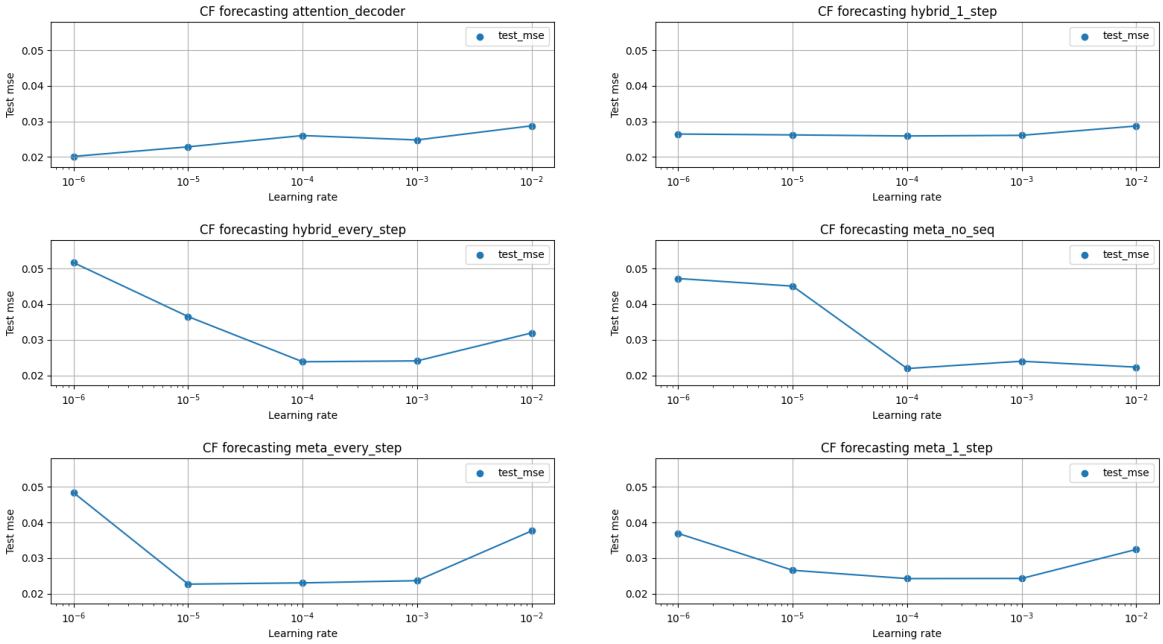

Figure 8.10: Test MSE per initial learning rate

8.6.2 Results CF forecasting per architecture

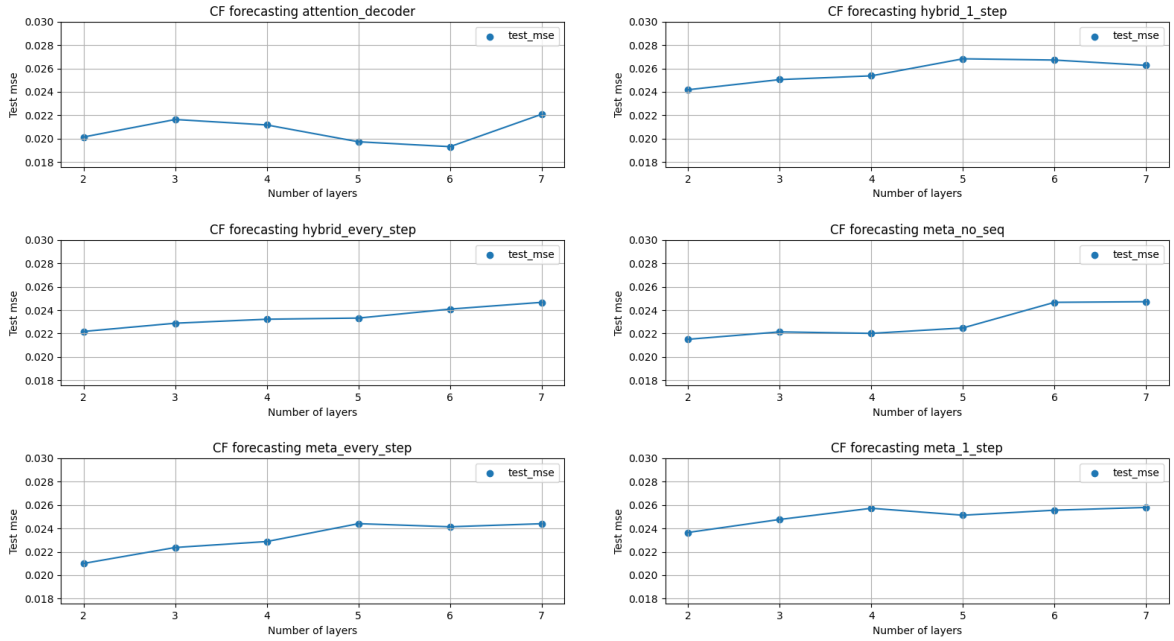

Figure 8.11: Test MSE per number of layers

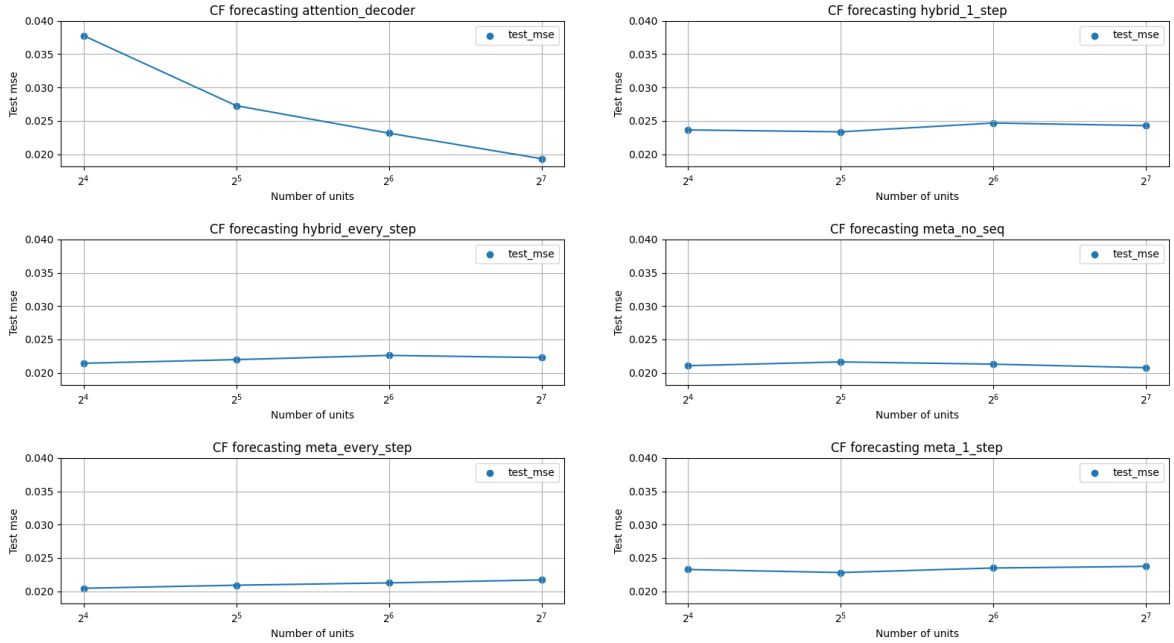

Figure 8.12: Test MSE per number of units

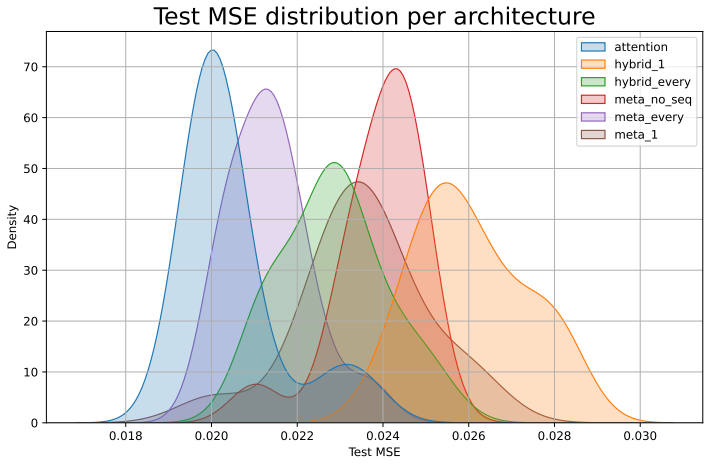

Figure 8.13: Density plots for test MSE values for CF forecasting experiments.

8.7 Cystic Fibrosis Improvement classification

8.7.1 Results from hyperparameter search

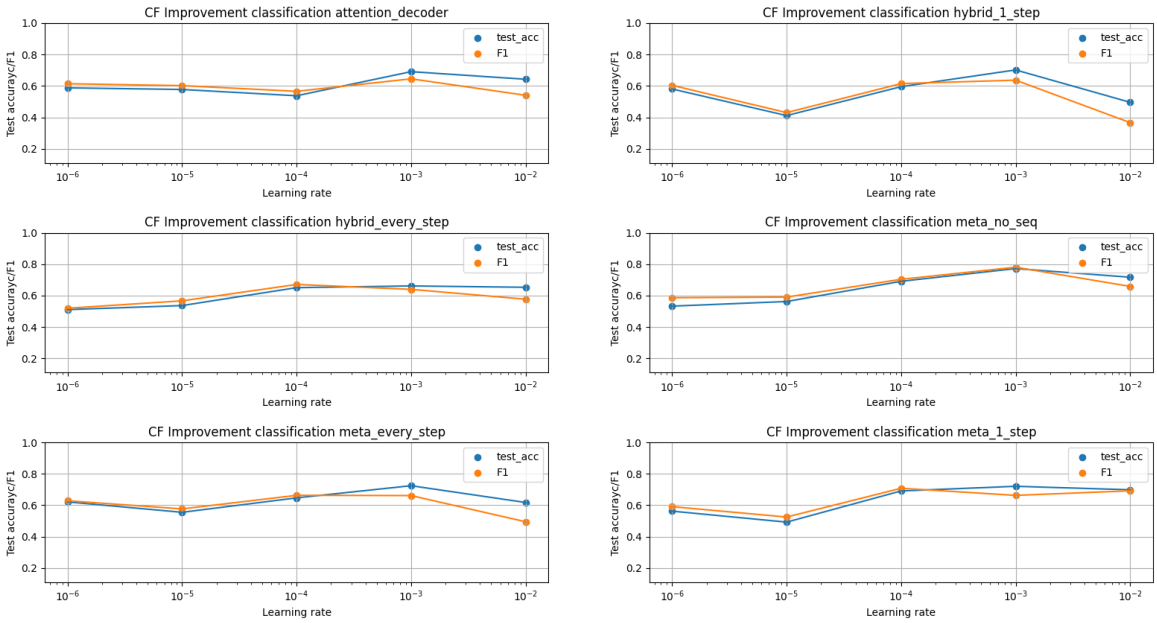

Figure 8.14: Test Accuracy and F1 score per initial learning rate

8.7.2 Results CF improvement classification per architecture

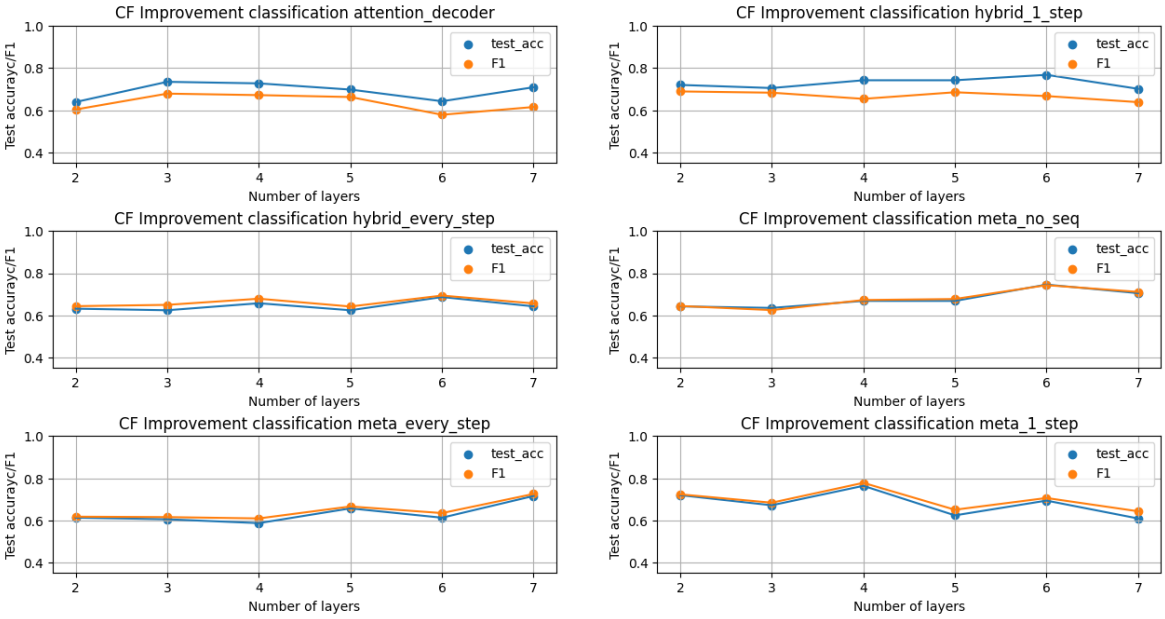

Figure 8.15: Test Accuracy and F1 score per number of layers

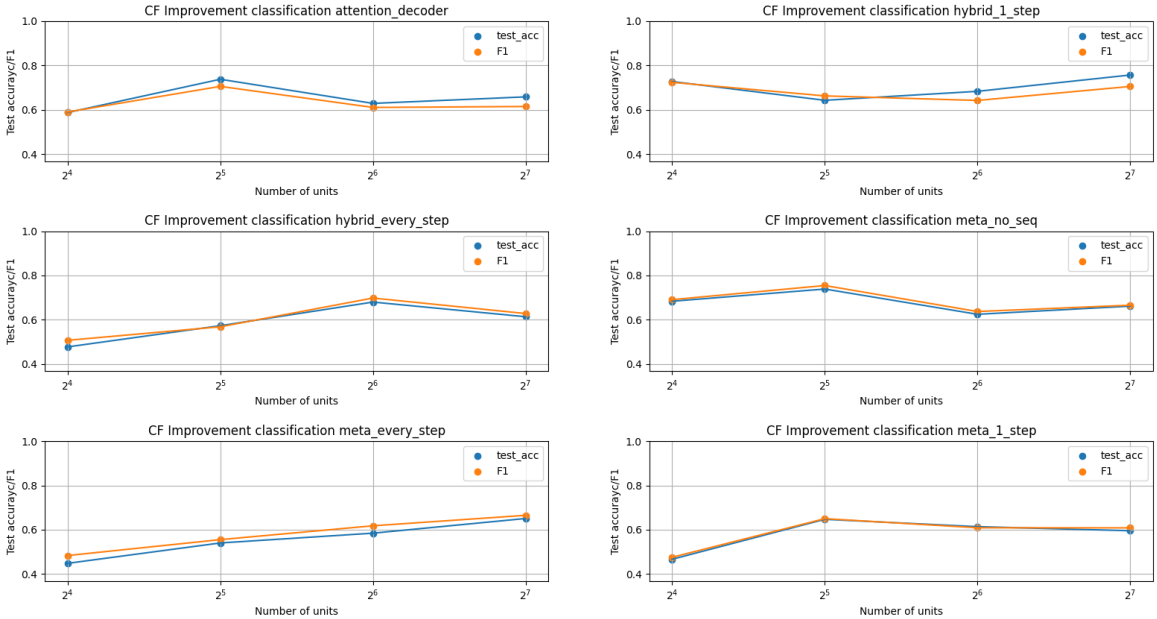

Figure 8.16: Test Accuracy and F1 score per number of units

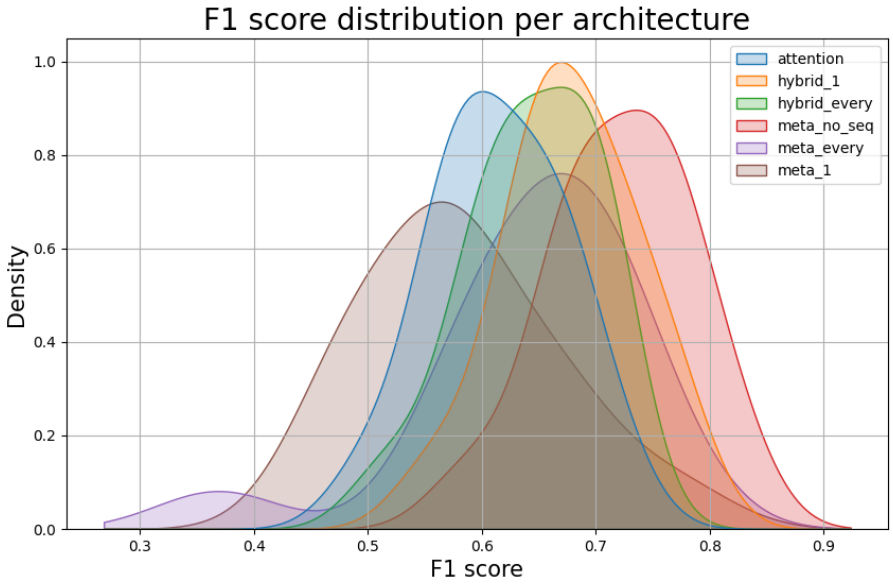

Figure 8.17: Density plots for F1 score for CF improvement classification.

List of Figures

List of Tables