Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Jacob Benedict Lindheim Belsvik

Machine Learning Based Digital Twins for Temperature and Power Dynamics of a Household

Master's thesis in Cybernetics and Robotics
Supervisor: Sebastien Gros
January 2023

NTNU
Norwegian University of
Science and Technology

Jacob Benedict Lindheim Belsvik

Machine Learning Based Digital Twins for Temperature and Power Dynamics of a Household

Master's thesis in Cybernetics and Robotics
Supervisor: Sebastien Gros
January 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Preface

This Master's Thesis is written in the fall of 2022 as part of my master's degree in *Cybernetics and Robotics* at the Norwegian University of Science and Technology (NTNU). It is the final product of my work in the course *TTK4900 - Engineering Cybernetics, Master's Thesis*.

This thesis investigates whether linear machine learning models can be used to model indoor temperature and power consumption of heating appliances. It is part of the larger PowAIoT project led by Sebastien Gros. In the PowAIoT project, cheap Internet of Things components for controlling heat pumps and measuring temperature and power consumption (among other data) are combined with a spot price-based Model Predictive Controller to control the reference temperature of the heat pumps. I hope my contribution to modeling temperature and power dynamics will help the PowAIoT project move forward with improved system models.

Sebastien Gros is the supervisor of this thesis, and I am grateful for our collaboration and all discussions on the topic, and for being available whenever I needed data insight or input on my work. I would also like to thank Dirk Peter Reinhardt for providing feedback on the report. Finally, I would like to thank all my fellow students and friends for making the last five and a half years unforgettable. Alas, five and a half years is too short a time to spend among such excellent and admirable people.

Abstract

Challenges with power production in Europe and the ever-increasing electrification of society have resulted in an increased cost of power consumption in the last couple of years. This has a significant impact on the household finances of the everyday consumer in Norway. Heating is a major part of residential power consumption, and utilizing smart control of heating could reduce power consumption drastically - especially during peak power spot prices. Model Predictive Control (MPC) is proven to be a good control scheme for indoor climate. The method exploits fluctuating spot prices to reduce power consumption bills whilst maintaining desired indoor temperatures. MPC schemes require good system models to simulate the indoor temperature and power consumption of heating appliances. Modeling these dynamics often require extensive simulations and data gathering.

In this Master's Thesis, pure machine learning methods are used as modeling tools for temperature and heat pump power consumption dynamics. The modeling is performed using data collected from a household in Trondheim that runs an MPC scheme for heat pump control. Direct linear and iterative one-step-ahead linear forecasting models are created to model the temperature dynamics. They are compared to the deep learning-based Temporal Fusion Transformer (TFT) model. All models are trained to give stochastic forecasts. The direct linear model outperforms the iterated one and shows similar performance as the TFT when trained on large data sets. The TFT outperforms the linear models when little data is available for training. The direct linear model shows great promise and manages to learn the temperature dynamics of the household quite well. Linear and deep learning-based prediction models are trained to give stochastic predictions for heat pump power consumption. The deep learning-based model consists of deep neural networks for each indoor heat pump unit. The linear models outperform the deep learning on little data, and their relative performance on longer data is quite similar. All proposed power consumption prediction models struggle to learn the true dynamics of the power consumption of heat pumps.

Sammendrag

Utfordringer med kraftproduksjonen i Europa og den stadig økende elektrifiseringen av samfunnet har resultert i økte kostnader ved strømforbruk de siste par årene. Dette har stor innvirkning på økonomien til husholdninger i Norge. Oppvarming utgjør en stor del av strømforbruket i boliger, og bruk av smart-styring av oppvarming kan redusere strømforbruket drastisk - spesielt når det er høy belastning på strømnettet, og strømprisene er høye. Modellbasert prediktiv kontroll (MPC) er bevist å være en god kontrollmetode for inneklima. Metoden utnytter svingende spotpriser til å redusere kostnadene ved oppvarming samtidig som en ønsket temperatur opprettholdes. MPC-metoder krever gode system-modeller til å simulere temperatur og strømforbruk for varmeapparater. Å lage disse modellene krever ofte omfattende simuleringer og datainnsamling, og er dermed en ressurskrevende metode.

I denne masteroppgaven brukes rene maskinlæringsmodeller for å modellere dynamikken til innendørs temperatur og varmepumpe-strømforbruk. Modelleringen er utført ved hjelp av data samlet inn fra en husstand i Trondheim som kjører en MPC-basert kontrollalgoritme for varmepumpestyring. Lineære direkte- og lineære iterative one-step-ahead-prediksjonsmodeller lages for å modellere temperaturdynamikken. De sammenlignes med den dyplærings-baserte modellen, Temporal Fusion Transformer (TFT). Alle modellene trenes til å gi stokastiske prediksjoner. Den lineære direkte modellen utkonkurrerer den itererative og viser lignende ytelse som TFT-en når den trenes på store datasett. TFT-en overgår de lineære modellene når lite data er tilgjengelig for trening. Den lineære direkte modellen er lovende og klarer å lære temperaturdynamikker i huset som reflekterer de sanne dynamikkene. Lineære og dyplærings-baserte prediksjonsmodeller trenes til å forutsi varmepumpens strømforbruk stokastisk. Den dyplærings-baserte modellen består av dype nevrale nett for hver varmepumpe. De lineære modellene utkonkurrerer dyplæringsmodellene på små datamengder, og deres relative ytelse på større data er ganske lik. Alle foreslåtte modeller for prediksjon av strømforbruk sliter med å lære den sanne dynamikken til varmepumpenes strømforbruk.

Table of Contents

List of Tables

List of Figures

Abbreviations

Abbreviation	Description
ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	AutoRegressive Integrated Moving Average
CPU	Central Processing Unit
DNN	Deep Neural Network
DSSM	Deep State-Space Models
DT	Digital Twin
GHI	Global Horizontal Irradiance
GPU	Graphics Processing Unit
IoT	Internet of Things
LSTM	Long Short Term Memory
MAD	Median Absolute Deviation
MAE	Mean Absolute Error
MET	Norwegian Meteorological Institute
MHE	Moving Horizon Estimation
ML	Machine Learning
MPC	Model Predictive Control
MQRNN	Multi-Horizon Quantile Recurrent Forecaster
MSE	Mean Square Error
OLS	Ordinary Least Squares
PSO	Particle Swarm Optimization
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Networks
SEKLIMA	Norwegian Centre for Climate Services
SGD	Stochastic Gradient Descent
SYSID	System Identification
TFT	Temporal Fusion Transformer
TRNSYS	Transient System Simulation Tool

1 Introduction

1.1 Background

In the last couple of years, the price of power consumption has increased drastically - especially in the winter [2]. The price of power consumption is based on supply and demand, which results in a fluctuating price over months and hours. For example, the spot price of power consumption often peaks between 8 and 9 in the morning and the late afternoon when households consume the most power. The energy production in Norway consists of mostly renewable hydropower, and the production highly depends on the weather. Norway is connected to the European power market through power cables, and the total electricity production and power consumption in the whole market affect prices in Norway. Many European countries currently struggle with electricity production. The cost of gas is historically high as a consequence of the war in Ukraine. Sweden, Finland, and France are struggling with their nuclear power plants, while Germany is planning to close their nuclear power plants. The wind turbines in Germany and Denmark produce more power than necessary with high wind, but too little when it is not windy. In December 2022, the mean hourly spot price for power consumption was 336 øre/kWh in southern Norway [3]. An analysis from Statnett states that the spot prices will decrease and stabilize at 50 øre/kWh in 2027 at the earliest [4]. We can therefore expect expensive and volatile power spot prices in the coming years.

A study from SSB shows that the increased power price has a great impact on households' finances - especially for low-income households [5]. Heating accounts for more than half of the power consumption in the residential sector [6]. Employing methods that reduce heating expenses would reduce the consumer's power consumption bills extensively. Reducing the heating during peak spot price periods would have the greatest impact. To do this effectively without manually monitoring the spot price and turning the heating on and off requires smart house solutions. This type of heating control has seen a rise in public popularity and interest. For example, Enova is an organization that finances companies and households with up to 10 000 NOK for installations of smart control of power consumption [7].

1.2 Smart House Solutions

Advanced control of indoor temperature and power consumption has been of great interest recently. Thermostats have been widely used to control indoor temperature and are a built-in feature on most modern heating appliances. Smart thermostats have emerged with additional functionality such as internet control and as a part of larger autonomous systems [8]. With the rise of Internet of Things (IoT), cheap components for temperature and power consumption measurements and appliance control have emerged. With smart systems such as Home Assistant, Apple HomeKit, and Google Home, consumers may use these components for manual and automated smart house control. Power suppliers are also entering this market. Tibber is a power supplier aiming to provide automation features that can reduce the power consumption of integrated IoT devices. These IoT devices often include some Application Programming Interface (API) that may be used to control and extract data from them. This provides a multitude of possibilities for custom and more advanced smart house control. Advanced control methods such as Model Predictive Control (MPC) have been explored for energy-efficient climate control [9]. MPC is a promising method for residential climate control that uses Digital Twins (DT) to simulate the indoor climate and heating power consumption for different sceneries. These simulations are used to optimize control inputs for the simulation period to achieve the desired indoor climate and power consumption. the MPC's performance relies heavily on the quality of the DTs of the temperature and power dynamics of the house.

1.2.1 Digital Twins

The term Digital Twin is often used to describe a digital replica of some physical dynamics or behavior. There are several types of DTs, ranging from a digital replica of a certain component or part to an entire system or process that may be utilized for simulation purposes. They are often used to increase the performance and reliability of their physical assets and to gain a greater understanding of complex systems.

For indoor climate dynamics, digital twins may be used to model the power consumption of air conditioning systems in addition to temperature dynamics. In this thesis, digital twins refer to models of temperature and power consumption dynamics. They provide insight into how indoor temperature and power consumption evolve with differing heating configurations and conditions. These digital twins, also referred to as system models, can be used in advanced control schemes to achieve desired indoor climate while limiting power consumption.

1.2.2 Challenges with Modeling Indoor Temperature

Several factors affect indoor temperatures. One of them is the outdoor temperature, but its impact depends on the insulation of the house and building materials. Solar radiation also has an effect. It provides heating during the day, but the size and orientation of windows determine the amount of sunlight entering the building and the effect of solar radiation. The heating and cooling appliances affect the dynamics of indoor temperatures significantly as well. Their positioning and efficiency affect the consistency of indoor temperatures. The behavior and number of occupants also have a great impact. Leaving doors open or closed affects the convection, and cooking food increases the temperature in the kitchen. Describing the effect of these factors qualitatively may be trivial, but modeling them quantitatively is challenging.

This makes creating models for indoor temperature dynamics often dependent on advanced modeling tools such as Transient System Simulation Tool (TRNSYS) [10]. Simulation-based methods need modeling expertise and extensive information about the relevant house which may be difficult to gather for the everyday consumer. Other modeling approaches require the collection of extensive data from a multitude of sensors placed in each room and outside. System Identification (SYSID) is one such method. In SYSID for temperature dynamics, thermal models are built to explain the convection and conductance between the air and structures. These models contain a multitude of unknown parameters which need to be learned using temperature measurements. SYSID models struggle to explain all aspects of heat transfer. For instance, they do not account for heat transfer to furniture, open doors, and random disturbances. The models are also house-specific, which means that model equations and parameters learned for one house may be completely wrong for another.

Mathematical models such as the ones described above are white box models where the parameter values explain the dynamics of the system. Machine Learning (ML) models have also been used for modeling temperature dynamics. These range from linear regression to deep learning algorithms. Machine learning methods are often black box models with limited explainability, but are easier to train than mathematical models. [11] explores using Long Short Term Memory (LSTM) networks for indoor temperature modeling. Hybrid methods combine mathematical models and machine learning to incorporate domain knowledge in the machine learning type models. Hybrid methods for greenhouse temperature modeling are explored in [12] and for a two-story house in [13].

Data collection over large time periods is often necessary to train both mathematical and machine learning-based models with decent results. When deploying a heating control system on a new household, you want it to work properly as soon as possible. If the control system requires training temperature dynamics models using collected data, it might take weeks before sufficient data is collected depending on the model. Creating models that achieve the performance necessary with limited data is an important research topic.

1.2.3 Challenges with Modeling Power Dynamics of Heat Pumps

Heat pumps are inherited complex systems. They use a combination of mechanical, electrical, and thermal processes to transfer heat. There is a wide range of factors that affects the performance of heat pumps. Heat pumps are generally more efficient at high temperatures, and less so at very cold or humid conditions. The efficiency of the compressor and other components in addition to the size and design of the heat pump also affect the power consumption. The operating condition influences the heat pump's heating requirements, and consequentially its power consumption. Larger houses with draughty windows and poor house insulation require more heating energy than small apartments with few windows and good insulation. This makes mathematical modeling of heat pump power consumption challenging, and exploring other methods such as machine learning approaches may be beneficial.

1.3 The PowAIoT Project

The work in this master's thesis is part of the PowAIoT (formerly POWIOT) collaboration project led by Sebastien Gros. The goal of this project is to create an IoT-based system for smart heating of a household. The system should be flexible and cheap to install with little engineering work. Expensive simulation methods such as TRNSYS should be avoided, and the system should work with (relatively) cheap IoT devices available commercially. The main research focuses in the project are data-driven and formal optimal control and modeling indoor temperature and the power dynamics of heat pumps. In previous work, an IoT-based solution has been designed to control a set of indoor heat pump units in four rooms in a house in Trondheim. A stochastic MPC scheme is used to minimize heating costs by exploiting the volatile power spot price while maintaining a desirable indoor temperature. Relevant data for the house is gathered continuously and is used to analyze the system and create models for temperature and power dynamics. Chapter 3 provides an overview of the components of this system. Right now, the system is only running on one house, but the system should be flexible enough to easily be installed and fitted to new households. Every house has different temperatures and power dynamics that need to be learned. The developed models should preferably be fitted to new households with as little data as possible. New proposed system models should be evaluated when trained on differing lengths of training data. The fewer data samples needed for performance convergence, the better.

The MPC scheme needs good models of indoor temperature and power dynamics to work properly. It optimizes the temperature and power dynamics according to some cost function for 12 hours ahead, and the temperature and power models should be able to simulate the dynamics for that period. In previous work, SYSID methods have been used on limited data to create these models with mixed results. As previously stated, SYSID methods may not account for all the temperature and power dynamics as a consequence of the complexity of the system. The SYSID models created on the house also lack some performance. It is therefore of great interest to research black-box machine learning methods and whether they can learn these dynamics.

In my project thesis, I explored using the state-of-the-art deep learning-based forecasting method Temporal Fusion Transformer (TFT) to model temperature and power dynamics. The learned temperature model showed great promise while the power consumption forecaster suffered from poor performance. The TFTs architecture is complex, which could present some challenges when used in the MPC scheme. The MPC scheme solves an optimization problem online on a Rasberry PI with limited computational resources, which limits the problem's desired complexity. The complexity of the system models has a great impact on the complexity of the optimization problem. Using deep learning techniques in the MPC scheme may increase computational time drastically. It is therefore of great interest to explore simpler machine learning methods such as linear regression analysis which limits the system model's complexity compared to deep learning models. If linear models achieve similar performance to that of state-of-the-art deep learning models, it would be beneficial to move forward using them as system models instead of the more complex ones. If the deep learning-based models outperform the linear models significantly, exploring methods to incorporate them into the MPC scheme would be of great interest.

1.4 Problem Description

This Master's Thesis is a continuation of the work in my project thesis. The overall goal is to create digital twins for indoor temperatures and the power dynamics of heat pumps that may be used in MPC optimization schemes. These models should be trained using data gathered from a real house. More specifically, this Thesis will focus on modeling temperature and heating power dynamics using linear regression analysis. The system models used in the PowAIoT-project need decent performance on limited data. The proposed linear models' performance when trained on increasing length of data is to be explored. As stated in section 1.3, simpler models should be used as system models unless more complex models achieve much better performance. These models will be compared to deep learning-based models trained on very limited data and on the full data set to study the differences in performance in these two cases.

With this as a basis, This Master's Thesis will consist of work on the following four tasks:

1. Developing machine learning-based linear models for indoor temperature and heat pump power dynamics.
2. Training the models on training data of lengths one week to one year to evaluate how training data lengths affect the models' relative performance.
3. Comparing the linear models with state-of-the-art deep learning models on limited and large data sets.
4. Analyzing the machine learning models to study their ability to capture the true dynamics of the system.

1.5 Scope and Limitations

The focus of this master's thesis is to explore pure ML methods for modeling temperature and power dynamics. There is a multitude of possible methods available for this purpose. Some different approaches are discussed, and two modeling techniques for both temperature and power consumption dynamics are chosen. These consists of linear regression-based models and state-of-the-art deep learning techniques. Other approaches such as hybrid methods and reinforcement learning-based MPC are outside the scope of this thesis. The models presented here are designed to be used in an MPC scheme. Implementing them and evaluating their performance in that setting is outside the scope of this thesis, and subject to future work. Some notes and discussions on their fit in an MPC scheme are provided.

1.6 Structure of the Thesis

This introductory chapter presented background information on the topic of the electricity market and how it affects the everyday consumer. Some smart house solutions were presented, and modeling temperature and power dynamics were discussed. Based on this, a problem description was formed. Chapter 2 presents a theoretical foundation on linear regression, deep learning techniques, and multi-horizon forecasting with some notes on model predictive control. This provides a foundation for the later presented modeling approaches. The following chapter introduces the system setup and provides information about the control scheme and data gathering. Chapter 4 presents linear and deep learning-based temperature dynamics models. They are evaluated and analyzed in the same chapter. Power consumption models for heat pumps are presented in chapter 5 with an evaluation and analysis. The results are discussed in chapter 6, and chapter 7 contains conclusionary remarks with suggestions for further work on the topic.

2

Machine Learning Theory

This chapter introduces machine learning methods relevant to this thesis. Some parts, especially the deep learning and time series forecasting sections, are heavily based on chapter 3 of my Project Thesis, resulting in some similarities.

2.1 General Machine Learning Problems

Machine Learning algorithms can be divided into several subcategories by different parameters. The type of experience used categorizes the algorithm into supervised/unsupervised categories. For example, if we have labeled data, supervised learning is used to create an input-output model, while unsupervised learning algorithms use unlabeled data to find patterns. There are also nuances of this categorization such as semi-supervised, online active learning, and reinforcement learning.

For supervised learning, a key aspect is the learning task at hand. Regression problems consist of predicting a numerical value y given a data point $\boldsymbol{x}$ while classification algorithms aim at specifying which of n categories a data point $\boldsymbol{x}$ is part of. This thesis consists of using regression models on a labeled dataset, so this class of machine learning is the focus going forward.

The performance measure of the ML model is dependent on the learning task. While classification tasks often use accuracy, regression tasks utilize Mean Square Error (MSE), Mean Absolute Error (MAE), or similar variants that compare model predictions with the true values. MSE is the most popular performance measure given by:

$$MSE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n} \mid\mid \boldsymbol{y} - \hat{\boldsymbol{y}} \mid\mid^2, \tag{2.1}$$

where $\hat{y}_i = f(\boldsymbol{x}_i)$ is the model prediction and n the number of samples. $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ contain all n true and predicted values. Whereas MSE aggregates the squared errors, MAE sums the absolute error. This means that large errors are not penalized as much when using MAE as shown in Figure 2.1. One has to assess the learning problem as a whole when deciding the performance measure to use, but because of its ability to penalize high errors, the MSE is often preferred over MAE.

Regression models are said to be deterministic when using MSE or MAE. To have a stochastic

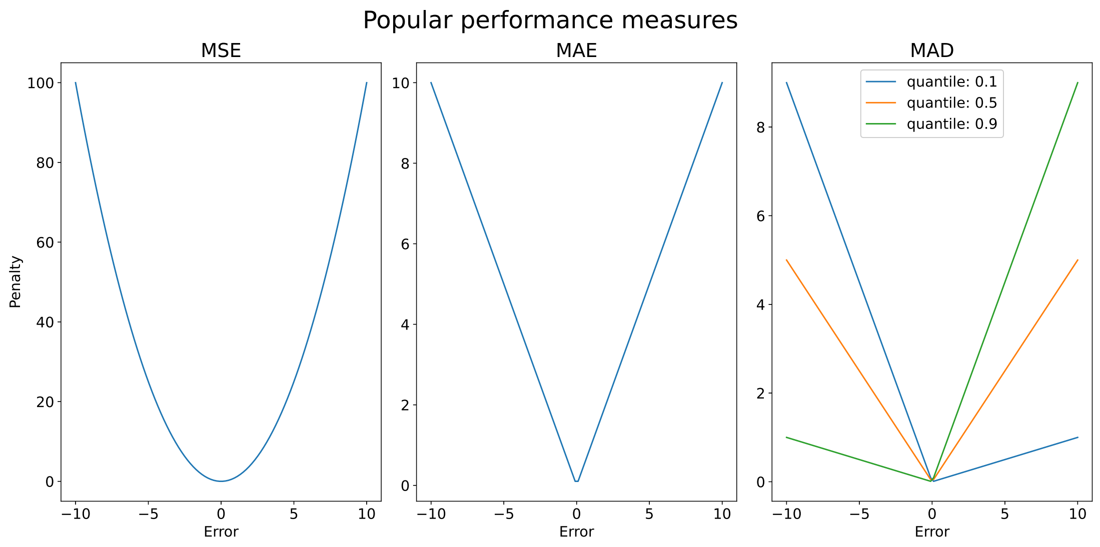

Figure 2.1: Popular performance measures.

regression model, another type of measure is needed, namely the Median Absolute Deviation (MAD), also called the quantile loss:

$$MAD(\boldsymbol{y}, \hat{\boldsymbol{y}}_q, q) = \frac{1}{n}\sum_{i=1}^{n} \rho(y_i - \hat{y}_{q,i}, q), \tag{2.2}$$

with $\hat{y}_{q,i} = f_q(\boldsymbol{x}_i)$ as the model prediction. $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}_q$ consist of all true and predicted data values for quantile q. $\rho(u, q)$ is given by:

$$\rho(u, q) = q \cdot \max(u, 0) + (1 - q) \cdot \max(-u, 0), \tag{2.3}$$

and gives asymmetric weights to the error based on the quantile q and the sign of error as shown in Figure 2.1. This formulation makes it possible to determine the spread of the true values $\boldsymbol{y}$, using several models $f_q(\boldsymbol{x}_i)$ with different quantiles.

2.2 Linear Prediction

Prediction is the act of using a model to predict the value of an unknown dependent scalar variable y (also called the target) based on a set of independent variables $\boldsymbol{x}$. Linear regression is a methodology for creating prediction models based on the linear relationship between the independent and dependent variables. For univariate dependent and independent variables, linear regression aims to fit a line to the data. With several independent variables, the linear regression model output is a weighted linear combination of the independent variables. This linear prediction model takes the form of

$$y_i = \boldsymbol{x}_i^\intercal \boldsymbol{a} + \epsilon_i, \tag{2.4}$$

for a data sample i where $\boldsymbol{a}$ consists of regression coefficients (also called model parameters) which denote an unknown slope and $\boldsymbol{\epsilon}$ some random measurement noise. For n observations, this model can be written in matrix form:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{a} + \boldsymbol{\epsilon}, \tag{2.5}$$

with $\boldsymbol{y}$ and $\boldsymbol{\epsilon}$ as $n \times 1$ vectors. X is a $n \times p$ matrix and $\boldsymbol{a}$ is a vector of shape $p \times 1$ where p is the number of independent variables. The coefficients of $\boldsymbol{a}$ need to be learned through regression analysis. For linear regression to work on a linear model for a data set, certain assumptions need to be fulfilled [14]:

1. **Linearity:** If a linear model can be used to describe the relationship between the variables, the relationship needs to be linear.
2. **Normality:** The residuals of the model needs to be normally distributed.
3. **Limited Multicollinearity:** The independent variables should be independent of each other. If there are dependencies between them, it would be difficult to assess the effects of each variable in the model.
4. **Homogeneity of variance**: The magnitude of the error should not differ significantly for different values of the independent variables.

The problem of finding the coefficients in the model is defined by minimizing some objective function L with respect to the model parameters $\boldsymbol{a}$:

$$\hat{\boldsymbol{a}} = \arg\min_{\boldsymbol{a}} L(\boldsymbol{a}) \tag{2.6}$$

If the assumptions above hold, the problem can be solved using Ordinary Least Squares (OLS). If they do not hold, but the problem is formulated in a way possible to solve using OLS, OLS can still produce reasonable results with varying reliability. OLS aims to minimize the sum of squared errors between the predicted values and the true values of the dependent variable with respect to $\boldsymbol{a}$ for n observations. In other words, OLS uses the MSE as an objective function L with $f(\boldsymbol{x}_i) = \boldsymbol{x}_i^\intercal \boldsymbol{a}$:

$$L(\boldsymbol{a}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^\intercal \boldsymbol{a})^2 = \frac{1}{n} \mid\mid \boldsymbol{y} - \boldsymbol{X}\boldsymbol{a} \mid\mid^2, \tag{2.7}$$

Under the assumption that the columns of $\boldsymbol{X}$ are linearly independent and that the system is overdetermined, the optimal unique solution to (2.6) with (2.7) as loss function is given by the OLS solution:

$$\hat{\boldsymbol{a}} = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y} \tag{2.8}$$

This system of equations has only one dependent variable, but OLS can also solve problems with several dependent variables. A linear model with M dependent variables contained in vector $\boldsymbol{y}_i$ for sample i and corresponding independent variables $\boldsymbol{x}_i$, model parameters $\boldsymbol{A}$ and some random noise $\mathbf{E}$ with n samples can be written in matrix form as:

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{A} + \mathbf{E}. \tag{2.9}$$

$\boldsymbol{Y}$ and $\mathbf{E}$ have shape $n \times M$. $\boldsymbol{X}$ has the shape $n \times p$ and $\boldsymbol{A}$ has shape $p \times M$. The corresponding OLS solution that minimizes the sum of squared errors with respect to $\boldsymbol{A}$, under the given assumptions, is given by:

$$\hat{\boldsymbol{A}} = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{Y}. \tag{2.10}$$

Deterministic linear models, such as the one presented above, predict the conditional mean of a target. Sometimes we also need information about the confidence of the predicted value of a model. Quantile regression is a method to achieve this. Quantile regression calculates a conditional percentile of the target where the 50th percentile is the median. The 50th percentile corresponds to the 0.5 quantile. A quantile model can also provide predictions at other percentiles. For example, there is a 90% chance the true value of the target is below the target prediction of the 90th percentile and a 10% chance for the 10th percentile. By this property, the assumptions for the linear regression are not needed for quantile regression, except for the linearity assumption when the underlying model is to be linear. It is often used when the assumptions are not satisfied [15]. A quantile regression model for a quantile q and a general linear model is given by:

$$y_q = \boldsymbol{x}^\mathsf{T}\boldsymbol{a_q} + \epsilon, \tag{2.11}$$

Quantile regression optimization aims to minimize the median absolute deviation. OLS minimizes the MSE, and cannot be used to solve quantile regression. Another type of optimization scheme must be used. Gradient descent optimization is a popular method for solving quantile regression problems.

2.3 Gradient Descent

Gradient descent is an optimization algorithm used to minimize an objective function $g(\boldsymbol{x})$. It is often the preferred method for solving quantile regression problems, and when the dependent and independent data have non-linear relationships and more complex non-linear models are used. When this is the case, OLS may only find a local minimum, while gradient descent can find the global minimum of the cost function. Gradient descent is an iterative method and is more computationally efficient than mathematical methods such as OLS for large data sets. Sometimes, there are physical restrictions on parameter dependencies in the model that OLS does not account for. OLS can be used to provide initial estimates for the model weights, and gradient descent can be combined with some method for constraining the model weights to find the optimal parameter weights. This is called projected gradient descent.

For machine learning problems, the objective function to be minimized is a loss function $L(\boldsymbol{W})$ where $\boldsymbol{W}$ denotes the weights of the general model $f(\boldsymbol{x})$. A training set with n samples consisting of independent variables $\boldsymbol{x}$ and labels $\boldsymbol{y}$ is used to train the model. For deterministic models, $L(\boldsymbol{W})$ is normally the MSE, while quantile regression utilizes the MAD. For a linear $f(x)$, $L(\boldsymbol{W})$ is convex, meaning that any local minima is a global minimum, and minimization is feasible. This is not the case for deep learning problems where gradient descent methods aim to minimize the loss function in a neighborhood.

The gradient descent algorithms update the model weights iteratively until some termination criteria are fulfilled. The weights are generally initialized randomly and updated using the gradient of the loss function with respect to the weights. The batch gradient descent algorithm calculates the error for all samples before the model parameters are updated by:

$$\boldsymbol{W}^{new} = \boldsymbol{W}^{old} - LR \cdot \nabla_{\boldsymbol{W}} L(\boldsymbol{W}), \tag{2.12}$$

Where LR denotes the learning rate of the algorithm. The weights should be updated until the global minima of $L(\boldsymbol{W})$ is reached, i.e when the magnitude of the gradient is sufficiently small. The learning rate is the main parameter that must be tuned for gradient descent. If it is too small, the algorithm will take very small steps toward the minima, and training the model may take a long time. If it is too large, the loss may not converge to the very minimum, and in some cases diverge.

While this method is computationally effective and generally guarantees a stable convergence for linear models, loss functions for deep learning problems are not convex because of the model architecture, and gradient descent may be trapped in a local minimum. Other variants of gradient descent are preferred for these learning tasks. The Stochastic Gradient Descent (SGD) method updates the weights based on the error for each training sample. For n training samples, the method updates the weights n times in each iteration. The gradient for each sample may not be a good estimate of the true gradient, but with enough randomness in the samples, the loss function will converge to the global minimum. The mini-batch stochastic gradient descent algorithm is a mixture of batch and stochastic gradient descent where the training set is divided into smaller batches. The algorithm updates the weights for each of these batches. This method combines the efficiency of batch gradient descent and the power of the stochastic gradient descent and is the preferred option for training deep neural networks.

Adam [16] is a recent extension to the mini-batch stochastic gradient descent method often used in deep learning. While SGD maintains a single learning rate, Adam uses adaptive learning rates for all parameter weights. It combines the advantages of two other improvements to SGD, namely the Adaptive Gradient Algorithm and the Root Mean Square Propagation, and solves deep learning problems more efficiently than other modern gradient descent algorithms [17].

2.4 Artificial Deep Neural Networks

Artificial Neural Networks (ANNs) are a class of ML algorithms that uses layers of nodes to approximate a model function $\boldsymbol{y} = f(\boldsymbol{x})$ with an independent variable vector $\boldsymbol{x}$ and a target vector $\boldsymbol{y}$. This function mapping can, with arbitrary accuracy, approximate any continuous function with the right amount of nodes and layers [18].

2.4.1 Architecture

Figure 2.2 presents the architecture of a feed-forward ANN. It consists of an input layer, an output layer, and a set of hidden layers. The number of nodes in the input layer represents the number of independent parameters (also called input parameters), and the output layer consists of target nodes. The number of hidden layers and nodes within each hidden layer is tuned to identify the optimal configuration for specific use cases. The network is called a Deep Neural Network (DNN) when several hidden layers are present. In a feed-forward DNN, the signal travels from the input nodes forward through the network. All nodes in a layer are connected to all the nodes in the adjacent layers, resulting in a dense architecture. The output of a node is a non-linear function of the weighted sum of the inputs in_i:

$$x_i \leftarrow g(in_i) = g\left(\sum_j w_{ji} x_j\right). \tag{2.13}$$

Here, w_i is the weights for the inputs to node i in layer l, and $j \in [0, n_{l-1}]$ where n_{l-1} is the number of nodes in the previous layer. g is called the activation function for the node and

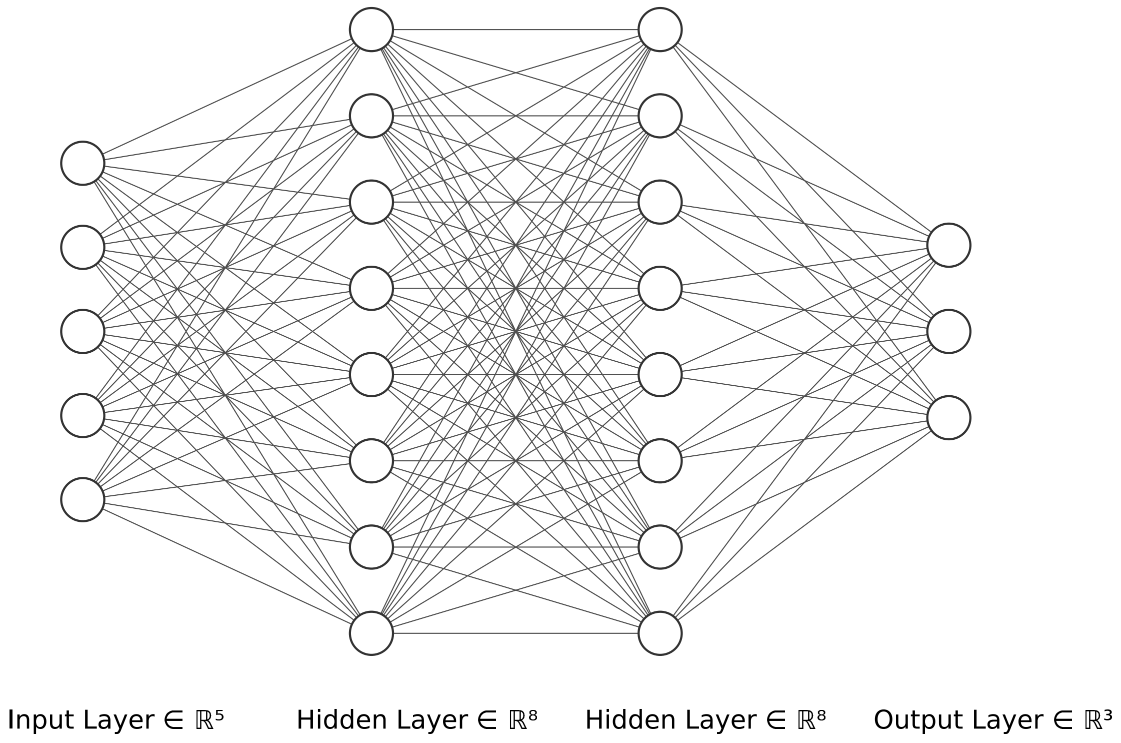

Figure 2.2: The architecture of an artificial neural network with two hidden layers.

decides if the neuron should be activated or not based on the input values. Different activation functions exist with different characteristics and benefits, and the chosen function for a DNN depends on the data and desired model. Popular activation functions include the sigmoid function, the Rectified Linear Unit (ReLU), and the tanh-function. These functions are demonstrated in Figure 2.3. Their advantages and disadvantages depend heavily on the training procedure and are discussed further at the end of the next section.

2.4.2 Training Procedures

Deep Neural Networks are trained using gradient descent methods. The inputs are propagated forward through the network to produce a scalar cost using a measure such as the MSE. For a stochastic model, a quantile loss based on the MAD can be utilized instead. This information is backpropagated to iteratively compute the cost gradient for each layer which is used to update the layer weights. Variations of mini-batch stochastic gradient descent are often used for training, with Adam being the most popular. These methods involve randomness in their operations. In addition, the network's parameter weights are often randomly initialized. As a consequence of these stochastic operations, training neural networks multiple times will yield different results. If the training procedure for the model is capable of converging to a global minimum, the differences will be minimal.

A challenge when training DNNs is preventing overfitting where the network is fitted to noise in the data and not necessarily the function to approximate. This reduces generalization, and the model may not perform well on new data. The main way to combat this is to make sure that the model complexity matches the complexity of the problem to be solved by tuning the num-

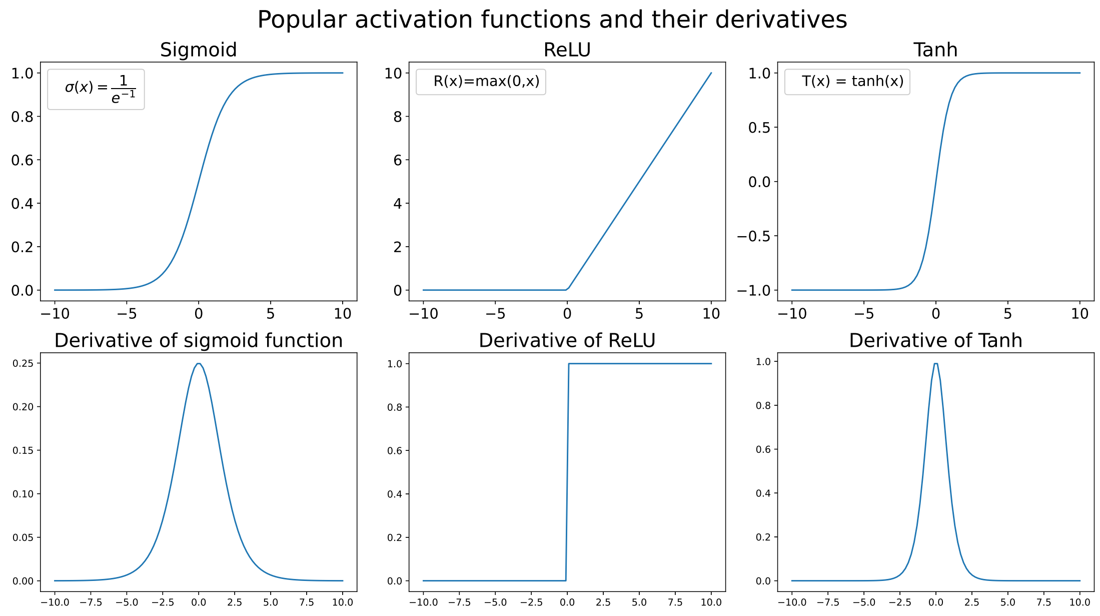

Figure 2.3: Popular activation functions for deep neural networks.

ber of layers and nodes. Tuning a model to avoid overfitting can be challenging, and is often time-consuming. There exists some regularization techniques used to prevent overfitting and increase generalizability. One such method is to add a regularization term to the cost function. L2 regularization is a method that penalizes large magnitudes of parameter weights and forces them to stay close to zero. With a loss function $LOSS$, weights $\boldsymbol{W}$ and L2 regularization, the cost function is as follows:

$$Cost = Loss + \lambda \cdot \sum_{w \in \boldsymbol{W}} ||w||^2, \tag{2.14}$$

where λ weights the L2 term and w denotes single weights in $\boldsymbol{W}$. This method reduces the model's complexity. If λ is too large, the network may not be able to create a model of desired complexity, and the model will be under-fitted. If a model is under-fitted, it struggles to match the complexity of the true dynamics. In addition to the learning rate and the number of nodes and layers, λ is an important tuning parameter. Other regularization methods include early stopping and the dropout algorithm [19].

Another challenge to training neural networks is the vanishing gradient problem. During backpropagation for very deep networks, the gradient may become vanishingly small, which prevents the weights from changing their value. This is often the result where the gradient of the activation functions has large plateaus close to zero. As Figure 2.3 shows, the gradients of the tanh and sigmoid functions have values very close to zero for high absolute values of the input. The ReLU only provides gradients equal to zero for negative inputs and a constant value for inputs larger than zero. This makes the ReLU a good choice for reducing the likelihood of the vanishing gradient problem.

2.4.3 Scaling Data for Deep Learning

Scaling the data is an important part of pre-processing for deep learning that is not necessarily needed for linear regression. The input and target variables all have different scales and distributions. The scaling of the weights learned by a deep neural network is affected by the scaling of the variables. As such, inputs with large values will have larger weights and thus be given more precedence by the model. In addition, models with large weights will have a higher sensitivity to input values and may become unstable. There are several ways of scaling the data, and here are the two most common practices:

- Scaling the data into a specified range (for example [0,1]) using min-max normalization. By using this method we retain the original distribution of the data except for the scaling. However, this method is highly sensitive to outliers and thus not very robust.
- Using Standardization (Z-score normalization). Here the data is scaled using the mean and variance of the data. This method is also sensitive to outliers, and there are no guarantees that the data is in a given range. Also, if the data does not have a Gaussian distribution, the scaled data will not have the same distribution as before the scaling.

When scaling the target variable, the method must be chosen together with the activation functions in the network. The ReLU and Sigmoid functions have an output range of zero to infinite and one respectively and should be used when min-max normalization is deployed. For targets of clear Gaussian distribution, the z-score normalization method should be used together with the tanh activation function - which ranges from -1 to 1.

2.5 Multi-Horizon Time Series Forecasting

A time series is a series of values sampled over time. Multi-horizon time series forecasting is the act of using past time series values of a target variable to forecast its values several time steps into the future, denoted by forecast horizons. The term forecast horizon has different meanings in different disciplines. In control theory, a horizon often denotes all future time steps from time t up to time $t + M$. In the ML field, each time step into the future is called a forecast horizon. With forecasts M time steps into the future, there are M forecast horizons. This is the definition used forward unless stated otherwise. A forecast differs from predictions by utilizing past time series values while predictions only utilize independent variables. In addition to past values of the *target*, several independent variables may also be utilized in the forecasting process. These independent variables could be categorical variables or time series that influence the target, How they are used in a forecasting model depends on the model. A general deterministic forecasting model with these inputs is illustrated in Figure 2.4. There are two main categories of time series forecasting

- **Iterated approaches** makes a one-step-ahead forecast and iteratively uses this forecast to make a forecast M time steps into the future.
- **Direct models** generate a forecast for all forecast horizons simultaneously. This can be done by creating individual models for each forecasting horizon or using sequence-to-sequence models that utilize a set of inputs to generate a set of forecasts.

For a theoretically fully correct model, the optimal forecast is given by the iterated one-step-ahead forecast [20]. Iterated models also consist of much fewer parameters than direct approaches making them more efficient. However, direct approaches are more robust to inefficiencies in the model and data making them more reliable for difficult forecasting problems. [20] shows that iterated forecast typically outperforms direct forecasts on macroeconomic time series, but the general performance of direct and iterated models heavily depends on the data

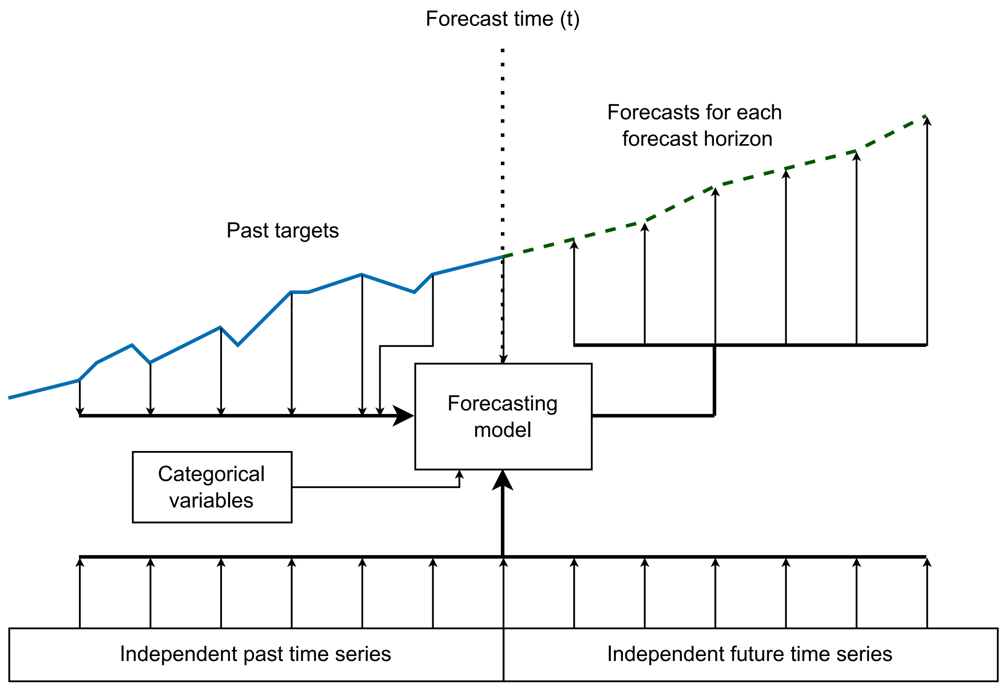
Forecast time (t)
Forecasts for each forecast horizon
Past targets
Forecasting model
Categorical variables
Independent past time series
Independent future time series

Figure 2.4: Illustration of a general forecasting model inspired by [1].

available.

Multi-horizon time series forecasting is an important topic of recent research in machine learning, with applications in quantitative economics, weather forecasting, medical applications, and similar methods used for machine translation and speech recognition. In classical machine learning approaches, methods using some weighted combination of past target variables are used. Auto-Regressive Integrated Moving Average (ARIMA) and Exponential Smoothing are two such models. There are several variations to these models. The main ARIMA model utilizes a moving average of the past differences in observations to create an auto-regressive model. Exponential smoothing also weighs the average of past observations, but recent values are given exponentially more importance than older values. In addition, Exponential Smoothing consists of error, trend, and seasonality components to build the model, and can be either additive or multiplicative. These models only utilize past observations, but extensions can be made to include independent variables in the forecasting.

2.5.1 Deep Learning Techniques for Multi-Horizon Forecasting

There are a variety of different deep learning algorithms for time series forecasting with different strengths and attributes. Many of these architectures are based on Recurrent Neural Networks (RNNs) and the Long Short Term Memory architecture. The following paragraphs contain an extract from my Project Thesis detailing information about RNNs and LSTMs.

The recurrent neural network is a type of neural network used to perform sequential tasks on time series data. This method is often used in machine translation and speech recognition tasks, but also for time series forecasting. RNNs utilize two ideas, namely parameter sharing and unrolling. In RNNs, the output of a node is fed recursively back into the node such that the output

y_t of the RNN cell is defined by:

$$h_{t+1} = g(w_x x_t + w_h h_t + b_h), \tag{2.15a}$$
$$y_t = g(w_y \cdot h_t + b_y). \tag{2.15b}$$

x_t is the input at time step *t*, h_t contains all values from previous states up to timestep *t*, b_h is the bias, $w_{x,h}$ are parameter weights to be learned and g is the activation function. The unrolling part of the RNN decides how many times the output is fed back into the node. I. e, if we unroll 3 times, we produce a network where the data goes through the process three times. This is shown in Figure 2.5. Here, the weights and biases in equation (2.15) are shared between each recurrent layer.

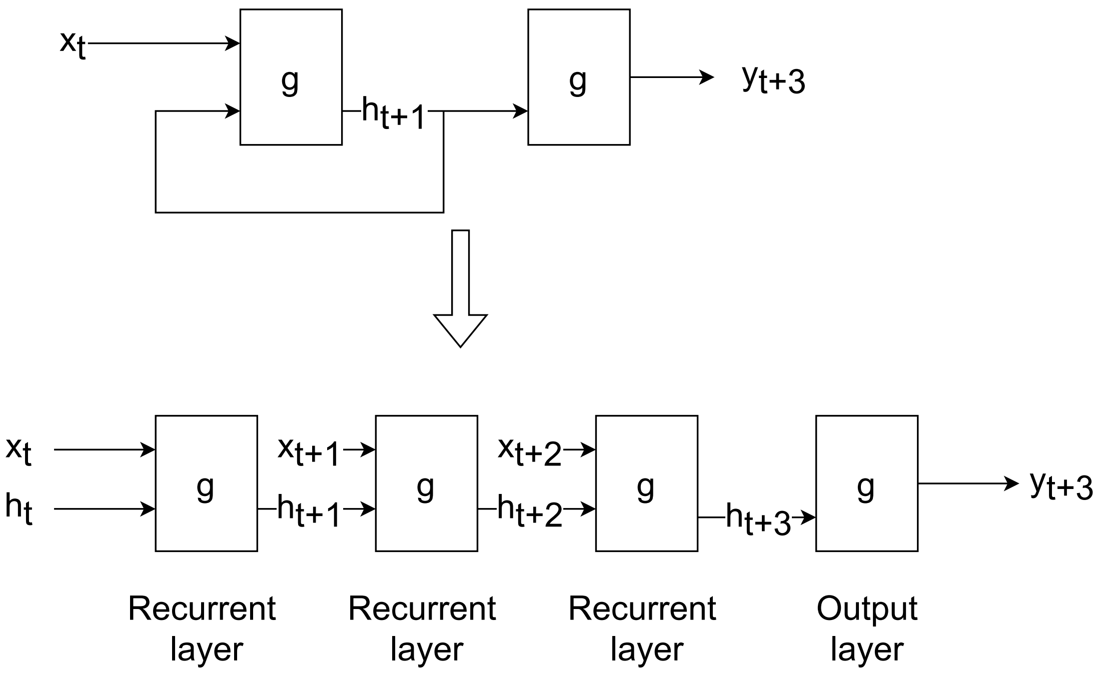

Figure 2.5: The architecture of an RNN cell and its unrolled version with 3 unrolls. This can be done an arbitrary number of times. g is an activation function, and each layer contains the weights and biases in equation (2.15).

RNNs are prone to exploding gradients (where large error gradients result in increasing model weights), but using the tanh activation function can prevent this. If the network is too deep, it may encounter a vanishing gradient problem.

Because of the vanishing gradient problem, basic RNNs cannot remember long-term dependencies, and we need more advanced types of RNNs. Long Short-Term Memory is a type of RNN designed to overcome this problem. The architecture of an LSTM cell consists of three *gates*; a *Forget gate*, an *Input gate*, and an *Output gate*. The forget gate is responsible for deciding whether the information from the previous timestamp can be forgotten or should be remembered. In the input gate, the cell learns the new information x_t, and finally, the output gate sends the learned information to the next timestamp. This is illustrated in Figure 2.6. The short-term memory in the LSTM is the hidden state h_t that consists of information about the current and previous timestamp. Long-term memory, on the other hand, is a name for the cell state c_t that contains information from all previous timestamps. Inside the LSTM architecture, it is normal that the tanh activation function is used to activate the cell state, while the sigmoid activation function is used for the output. While the LSTM can learn longer dependencies than the standard RNN, there is still a limit to how far into the past it can look. It is popular to use the LSTM in encoder-decoder architectures. The encoder-decoder LSTM is often used in sequence-to-sequence problems (such as machine translation) where the aim is to forecast a sequence of time series data based on a previous set of time series data. The encoder consists of an LSTM node that reads and summarizes the information from the input, and the decoder (also an LSTM node) outputs the entire forecast.

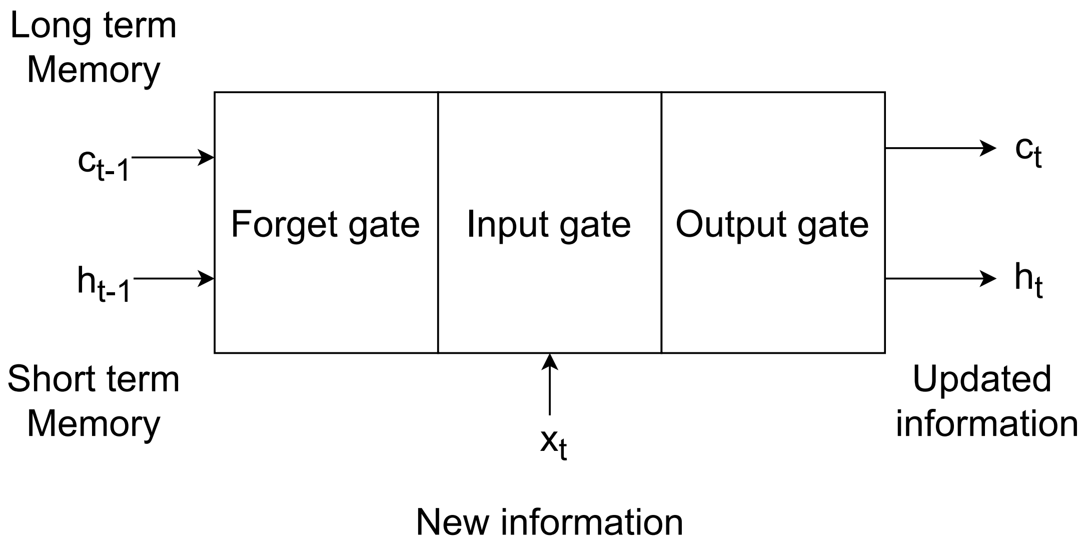

Figure 2.6: Overview of an LSTM cell with hidden state h_t, cell state c_t. and input x_t.

In recent years, there has been extensive research on deep learning-based multi-horizon forecasting. The following models are among the most popular state-of-the-art deep learning architectures:

- Amazons DeepAR [21] is an autoregressive model that utilizes known future independent variables in the system and can support using several time series by creating one global model instead of one per time series. DeepAR is an iterated approach that makes one-step-ahead forecasts with Gaussian distribution.
- Deep State-Space Models (DSSMs) as proposed in [22] combine deep learning with state-space models to describe a sequence of data. These normally consist of a (Markovian) transition model and an observation model which are both typically nonlinear. DSSM models enable the use of some pre-existing knowledge about the system, while the exact dynamics are unknown. It is standard for DSSMs to use RNNs to support Bayesian filtering or smoothing, while other approaches, such as in [23], use Extended Kalman filters in the optimization framework. As with DeepAR, DSSM methods are iterative, and all future independent variables must be known at forecasting time.
- A Multi-Horizon Quantile Recurrent Forecaster (MQRNN) is proposed in [24]. This forecaster utilizes a sequence-to-sequence neural network with recurrent and convolutional structures and quantile regression. This is a direct method that can account for time-dependent and static independent variables, multiple time series, seasonality changes, and future planned events.

- The Temporal Fusion Transformer [1] is a state-of-the-art Transformer-type forecasting algorithm from Google that learns temporal patterns in historical data to enhance relevant input data during training. It is a direct method that uses recurrent layers for local processing while long-term dependencies are learned using self-attention layers. With gating mechanisms, it can adaptly use simpler mappings where more complex non-linear mappings are not needed.

Deep learning-based multi-horizon forecasters generally utilize complex structures with potentially thousands of parameters with nonlinear dependencies to be learned. With large datasets, training deep learning architectures may take hours or days whereas training simpler linear models take minutes. Deciding on which type of model architecture is necessary should be a combination of model application and data set in addition to time and resource requirements.

2.6 Model Predictive Control

Model Predictive Control is a control scheme that aims to optimize the behavior of a system from control time t over a time horizon $t + T$. In contrast to ML, a time horizon in MPC theory refers to all time steps in $[t, t + T]$. Digital twins, or system models, are used to simulate the behavior of the system for the optimization period. The goal of the MPC algorithm is to find the set of optimal controllable input parameters that gives the optimal trajectory of the system states based on certain requirements. The optimal trajectory is typically the one that minimizes the expected deviations from the desired state values. The MPC takes into account the likelihood of different errors occurring and tries to reduce the errors that are most likely to occur. An MPC scheme consists of:

- A system model which represents the dynamics of the process. The model is often represented as a constraint in the optimization problem.
- A cost function J which is to be minimised. The cost function represents the desired behavior of the system. It is often a sum of squared errors between the reference variables and their respective state estimates over the whole time horizon.
- Other physical constraints such as limitations on the state and input parameters.
- A numerical optimization algorithm that minimizes the cost function J using some control input.

Combining these components results in a quadratic programming optimization problem. The MPC scheme runs iteratively during process control at a certain rate. At the time t, the current system states are sampled, and an optimization algorithm is used to determine a set of controllable parameters that minimize the cost function over the time horizon. The optimal control parameter values for this time step are deployed on the system.

There are several improvements and variations to this vanilla MPC scheme. Nonlinear MPC, explicit MPC, Robust MPC, and Stochastic MPC are some of these. They all have different strengths, weaknesses, and use cases, but their characteristics are outside the scope of this thesis.

2.6.1 System Models in MPC Optimization

In MPC optimization, system models are used to predict or forecast state trajectories into the future. Linear MPC methods require these models to be convex and twice differentiable as the optimization problem is often solved using gradient-based methods such as the interior point method [25]. This makes linear system models desirable in MPC formulations. However, ANN-based MPC methods have been explored in the literature with good results [26]. Deep learning techniques are generally more complex than linear models and require much more computational power than simpler methods. Since their use makes the problem nonlinear, nonlinear MPC optimization methods are needed which poses challenges for stability and numerical solutions. To avoid gradient calculations, one could deploy stochastic optimization methods such as Particle Swarm Optimization (PSO) [27]. This is a stochastic search technique that iteratively searches for the global minimum of a given cost function using particle swarms. However, there is no solid mathematical foundation for these types of models, and they often require longer computation time than derivative-based methods.

In the traditional MPC formulation, deterministic system models are used to predict the state trajectories. However, most systems have some stochastic uncertainty which deterministic models fail to account for. Including information about the expected error in an MPC scheme ensures adherence to the constraints. This information may be included by using stochastic system models with quantile outputs.

Model predictive control optimization is computationally expensive, but there exist methods for improving the runtime. Sparse system models can be exploited in condensed-based solvers such as the qpOASES from [28]. In this quadratic programming solver, condensing can be used to identify and remove inactive constraints from the problem. Using solvers such as qpOASES to solve MPC problems may increase efficiency drastically if system models have an exploitable structure.

3

System Setup

This chapter introduces the system setup on a house in Trondheim which provides a basis for this Master's Thesis. It is heavily based on chapter 2 of my Project Thesis, resulting in some similarities. The system has been active since the spring of 2021 and is constantly collecting data.

3.1 Components for System Control

This section describes the physical components and data sources of the heating control algorithm used in the Smart House.

3.1.1 Heating

The heating in the four rooms *main, living, livingdown* and *studio* is performed using heat pumps. The heat pump system consists of two outdoor units and one indoor unit in each room. The *living* indoor unit is connected to one outside unit, and the rest of the indoor units are connected to the other one. The indoor units are independent of each other so that each room can have a different temperature profile. The indoor units are from now on referenced as the heat pumps as it is these units that can be controlled. Table 3.1 gives an overview of the possible controllable states of each heat pump. The heat pumps are connected to a circuit independent from all other appliances in the house. This provides the possibility of measuring the total power consumption of the heat pumps at all times, but not the individual heat pumps. At some times, heavy machinery has been connected to the circuit for short periods resulting in spikes in the measured power consumption.

The room layout in the house has an effect on the convection in the house. *studio* and *livingdown* are situated downstairs. A staircase connects these rooms to *main* upstairs which is directly connected to *living*. Therefore there is a large convection between *studio* and *livingdown*, and *main* and *living*. The heating in the two rooms downstairs should also have a greater impact on the temperature upstairs than vice versa as warm air tends to rise. Figure 3.1 illustrates the direct room connections. The temperature in *living* and *main* are most influenced by outside temperature and solar radiation.

Table 3.1: Possible states for each indoor heat pump unit.

Setting	Possible states
On	True, False
Set (target temperature)	Integer $\in [16°C, 32°C]$
Fan(level)	quiet, low, medium, medium_high, high

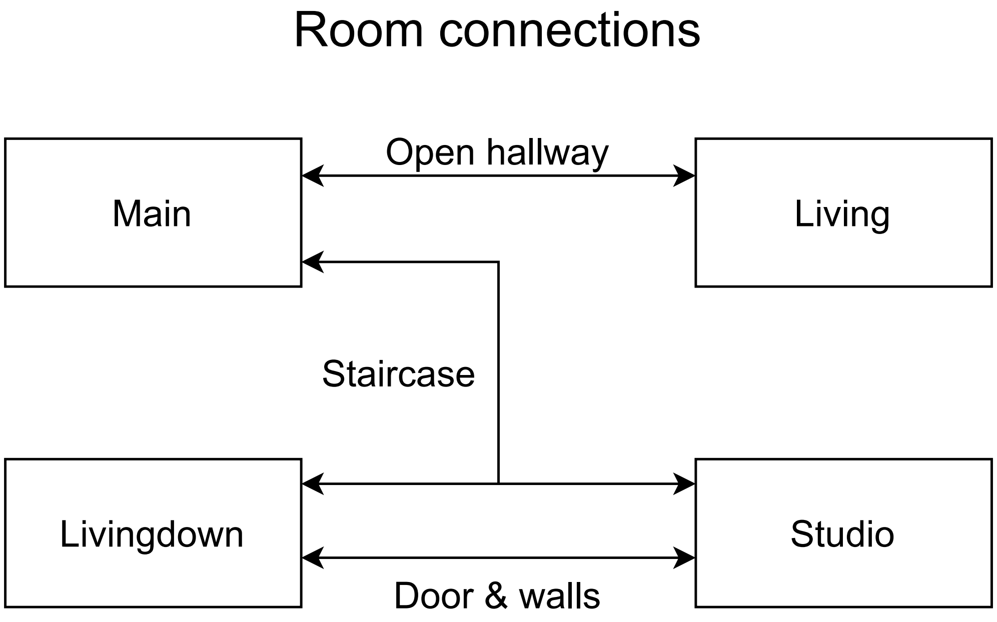

Figure 3.1: Illustration of the room connections.

3.1.2 Sensibo - Communication with the Heat Pumps

For each indoor heat pump unit, there is installed a Sensibo Sky device [29]. These devices communicate with the heat pumps using infrared signals and may extract and control the states of the connected heat pump. Sensibo Sky offers Wi-Fi-based control of heat pumps through a mobile application or an API in addition to temperature and humidity measurements.

3.1.3 Power Supplier

Tibber [30] is used as the power supplier. Tibber is a power supplier that focuses on smart control of electrical appliances for reduced power consumption. As such, they provide a variety of IoT services. Tibber Pulse is a device that can be connected to the power meter of a house to provide real-time power consumption measurements through the Tibber API and mobile application. Using this device, the total power consumption of appliances connected to the two circuits in the Smart House is measured at hourly intervals in real time. Power usage over 5 minutes is also measured. Figure 3.2 presents the total power consumed by the heat pumps for a whole year and for two days in addition to its distribution. Note the heavy oscillations for the power consumption over five minutes intervals.

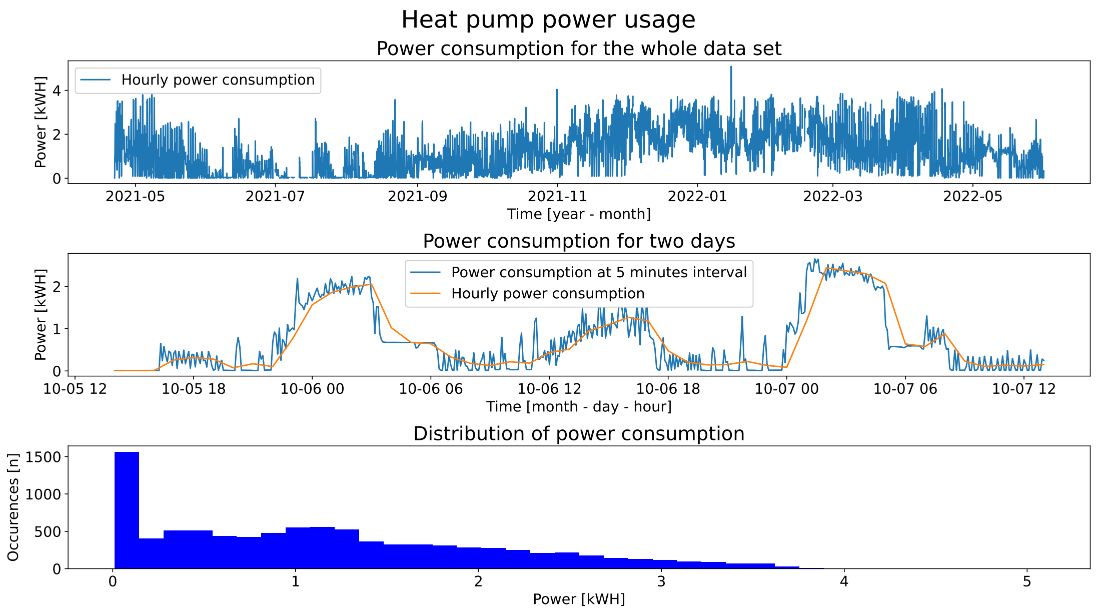

Figure 3.2: Measured heat pump power usage for a year and two days in addition to its distribution.

3.1.4 Spot Price

In the European power market, the spot price for power is given at an hourly rate. As the supply and demand of power vary with seasons, and over single days, the hourly price for power varies as well [31]. These variations are key for saving power, and spot price data from Nord Pools API is used in this system control setup. Nord Pool is a European power trading platform that calculates the hourly spot price in the European markets daily based on the purchase and sale of electricity across the continent.

3.1.5 Weather Data

The weather has a substantial effect on indoor temperatures and is thus important to consider in indoor heating control. Several services provide weather data - both historical and forecasts:

- **Norwegian Meteorological Institute** (MET) provides an API where historical measured weather and weather forecasts can be accessed and extracted.
- **Norwegian Centre for Climate Services** (SEKLIMA) provides in collaboration with MET a website service where historical weather and solar data from weather stations in Norway is available for download.
- **Solcast** is a global provider of weather and solar data. Through their API, weather forecasts and historical data for desired coordinates can be accessed.

There are some limitations to downloads from SEKLIMA, and downloading all historical data needed with the desired resolution is not currently possible. The data from MET have some gaps and are generally found to be noisier than the data from Solcast. In addition, MET does not have solar radiation data easily available. Therefore, air temperature and solar Global Horizontal Irradiance (GHI) from Solcast is currently used for house control and is chosen for this thesis as well. They are presented for the whole data set in Figure 3.3.

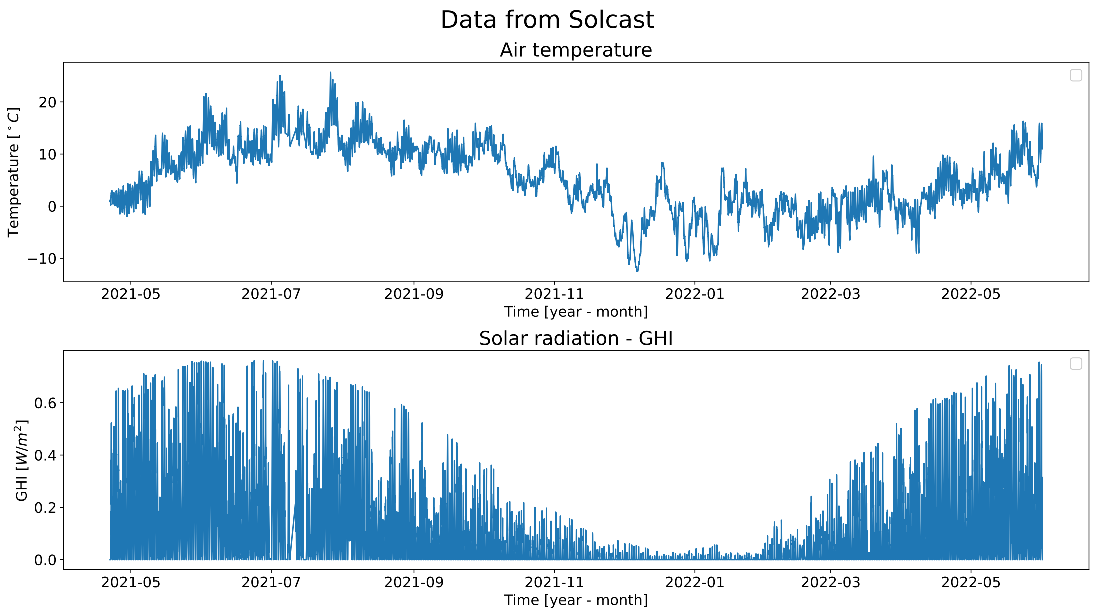

Figure 3.3: Temperature and radiation data from Solcast.

3.1.6 Control Unit

A Rasberry Pi is used for temperature control and data collection. The heat pump control scheme runs on the Rasberry Pi approximately every five minutes. It consists of two parts for setting the heat pump settings at each time step:

1. A Moving Horizon Estimation (MHE) method is used online to estimate the current system state from N past measurements and the system model. The estimation is performed by minimizing the discrepancy between the measured data and model predictions over a moving horizon. This method reduces the effect of noise from the sensor measurements. The state estimates are fed into the MPC as initial conditions.

2. The MPC solves the finite horizon open-loop optimal control problem for a horizon of approximately twelve hours. The states of each heat pump are used as independent control states, with weather data and spot price as other independent (non-controllable) states. The cost function in the MPC consists of terms for comfortable room temperatures (with hard limits) and the total price based on power consumption and spot prices. The optimization algorithm calculates a set of heat pump settings that minimize the cost function for a period of twelve hours into the future with time steps of five minutes using models for temperature and power dynamics as constraints. The final step consists of sending the optimal heat pump settings for the first time step to the heat pumps.

The data used for control is extracted from the Sensibo and Tibber devices in addition to the Solcast API through WiFi. All data is saved remotely for analysis and modeling. The sample periods of the different data measurements are provided in Table 3.2.

Table 3.2: Data and measurements.

Data	Provider	Sample period
Temperature in each room	Sensibo	5 sec
Power consumption	Tibber	1 h & 5 min
Outside temperature	Solcast	5 min
Solar radiation (GHI)	Solcast	5 min

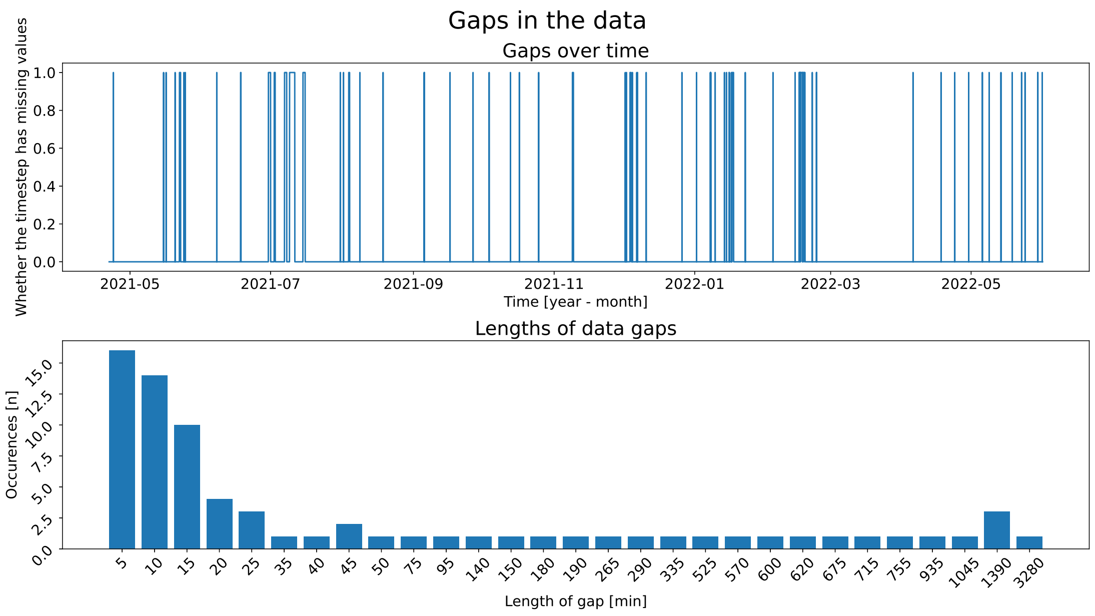

Figure 3.4: Gaps in the collected data.

3.2 Data Collection

Data is collected for three purposes; an online optimization of the power consumption by controlling the heat pump settings, analyzing the data, and creating models for temperature and power dynamics. Preserving high data quality and sanitizing the data are important for the system to run smoothly. During the early stages of deployment, the system struggled with some robustness in the data collection scheme, resulting in gaps in the data. The data collected in the early spring of 2021 is omitted in this thesis, but there are still some gaps in the data used. Most of these gaps are small (under one hour), but for some periods, hours of data are missing. This is shown in Figure 3.4. One could argue that interpolating the data in the shorter gaps would not introduce untrue dynamics in the system, but this is not the case for long gaps. For short data sets, interpolation could be necessary to create decent models, but to avoid introducing possible untrue dynamics in the data, the data is kept as is. From Table 3.2, we see that the data from different sources are sampled at different periods. In this system, where the heat pump settings are updated every 5 minutes, a higher sampling frequency is not necessary, so the data from Sensibo is resampled into 5 minutes intervals. In addition, we have to make the non-number inputs numeric since ML methods do not support text inputs. Thus, the following conversions are made:

- The on/off-setting for the heat pumps has two values true, false, and these values are mapped to $\{0, 1\}$.

- The fan level-setting for the heat pumps is given the following mapping: $\{$quiet, low, medium, medium_high, high$\} \mapsto \{0,1,2,3,4,5\}$.

As stated in 3.1.1, heavy machinery is at some times connected to the heat pump circuit. During the data gathering period, there have also been made renovations on the house where for example windows have been changed in the middle of the winter. These phenomena introduce dynamics that the measured data is unable to explain and reduce the quality of some data. During online optimization, the MPC should account for these phenomena, but their impact on fitting models to the data may be non-trivial. One could argue that it would be beneficial to remove the anomalies from the data to reduce their effect on training system models. In real-world scenarios, system models should be able to handle these anomalies. To test the robustness of proposed system models, this data is also included in the data sets.

Data inspection is a useful tool to validate and explore patterns in the data. Figure 3.5 presents the heat pump setting, room temperature, outdoor temperature, and total power consumption over two days. It illustrates the temporal dynamics of indoor temperatures. Some possible correlations between the heat pump settings and the temperatures and power consumption are also apparent. To further investigate the parameter distributions, a distribution plot using the parameters for the main room is constructed in Figure 3.6. This figure also presents the joint distribution between the parameters. Only the indoor and outdoor temperatures have a somewhat Gaussian distribution. The GHI has most of its values very close to zero. There is a clear relationship between the temperature in *main* and the outside temperature. As the heat pump settings are mostly discrete, there is difficult to describe the exact relationship between the variables using this plot. Figure 3.7 presents a correlation matrix for all measured parameters. Blue squares suggest a positive correlation, while red squares represent a negative correlation. From this plot, it is clear that there is a strong positive correlation between indoor temperatures. The indoor temperatures also have a strong correlation with the outside temperatures. *main* is most affected, and *studio* the least. This is consistent with the layout of the house. Power consumption has a negative correlation with all indoor temperatures, outside temperatures, and GHI. It has some positive correlation with the heat pump settings except for *Set* in *livingdown*. There is also some negative correlation between several of the heat pump settings and the outside temperature and GHI.

3.3 Data for Training and Testing System Models

The collected data from the house used in this thesis spans the period of 21.04.2021 to 31.05.2022. The last 10% of this data is set aside as a test set to evaluate the proposed models. The test set consists of data from 22.04.2022 to 31.05.2022. In this period, the outside air temperature ranges from $-1°C$ to $16.3°C$ and the power consumption is from 0.01 kWh to 3.485 kWh. While these ranges of data do not capture all seasons perfectly with any of the extremes, the test data still represents a large portion of the yearly data. The rest of the data is used as training sets.

In real-world scenarios, trained models are utilized on subsequent data and often retrained on new (and old) data at some interval. All training sets used to train models in this thesis are directly set before the test set. The longer the training data sets, the more data into the past is used. In this thesis, training sets ranging from one to fifty weeks are utilized. The length of the training set used for different models is specified when relevant.

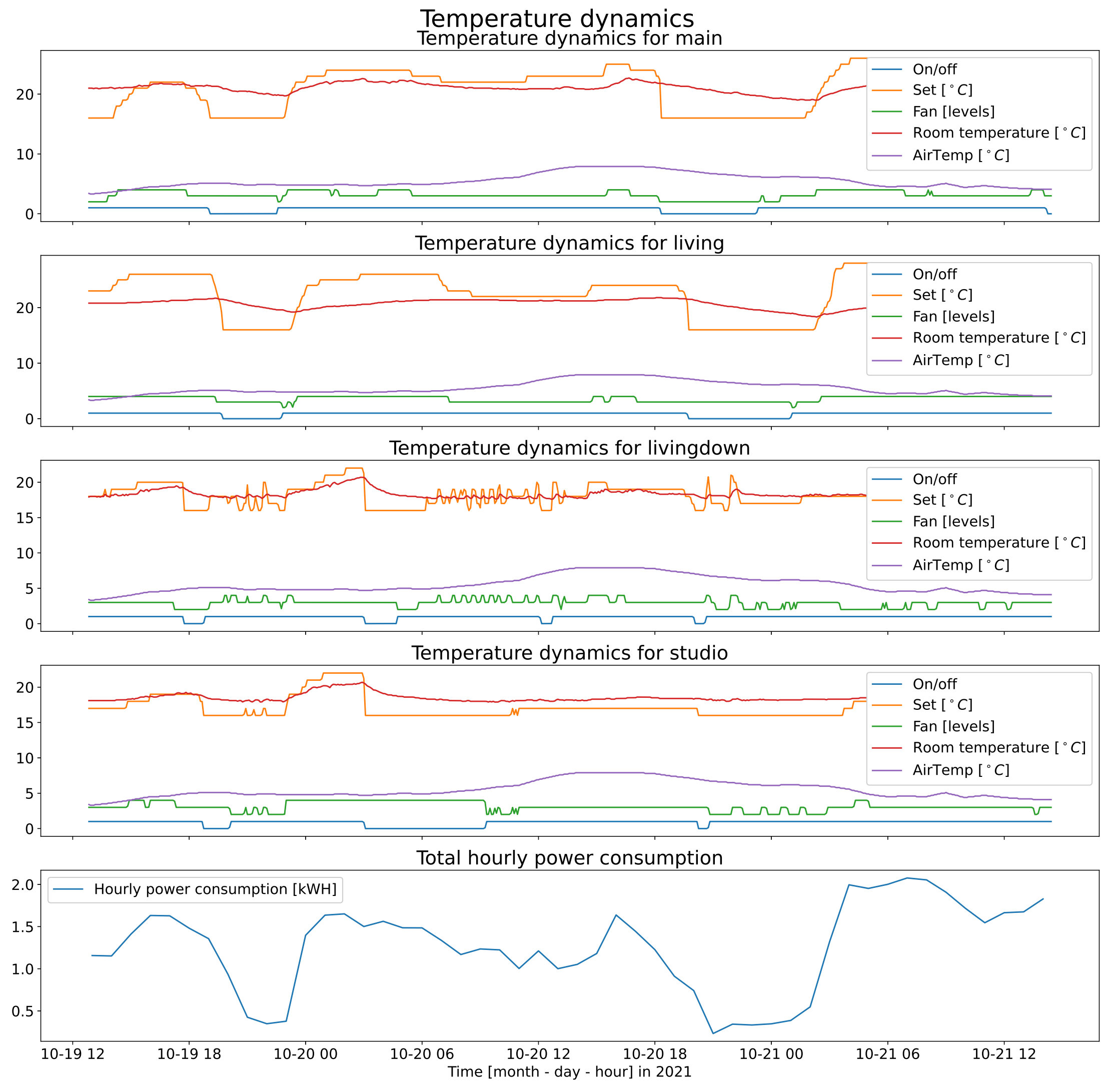

Figure 3.5: Temperature and power dynamics for the rooms over two days.

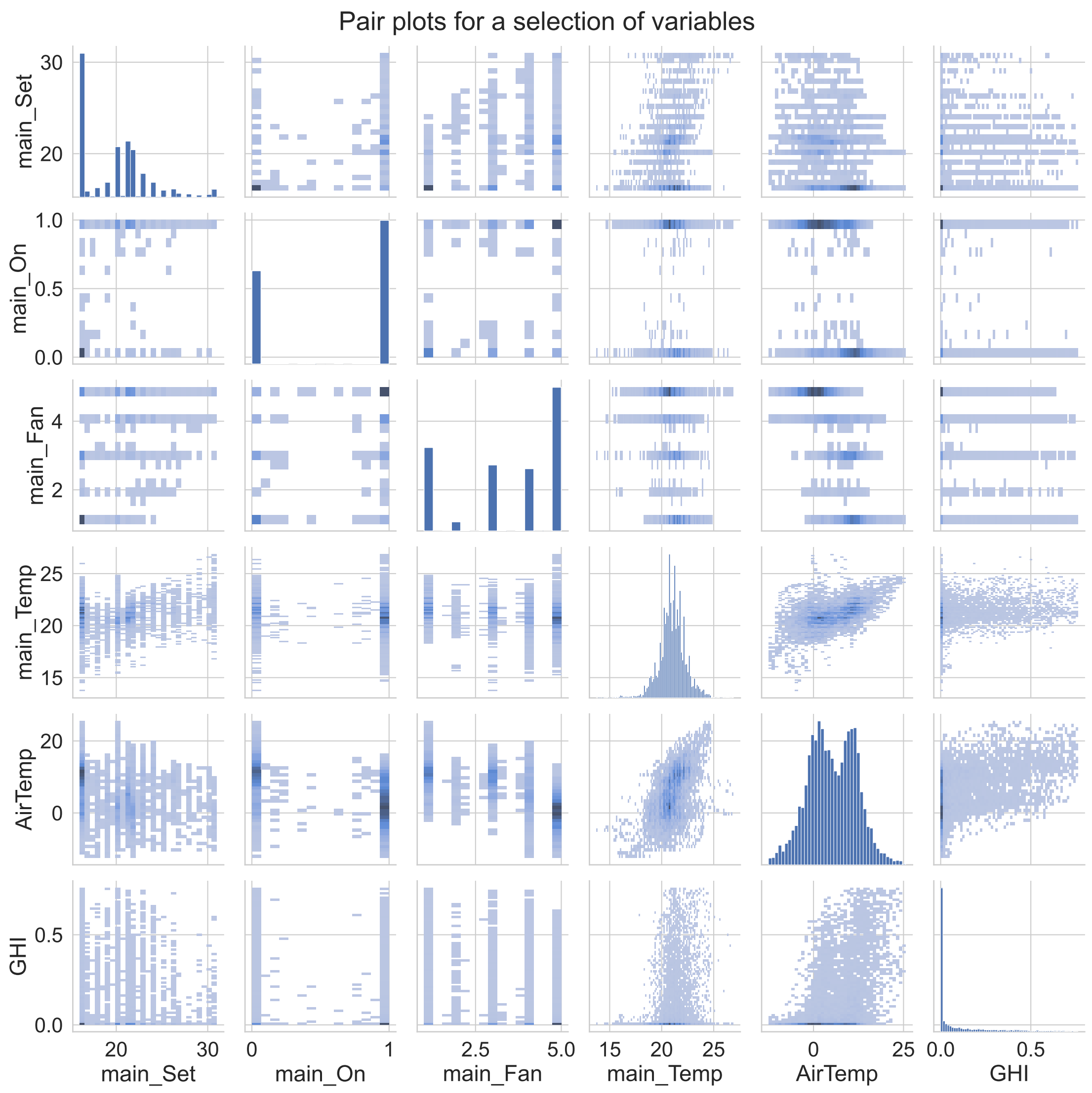

Figure 3.6: Parameter distributions.

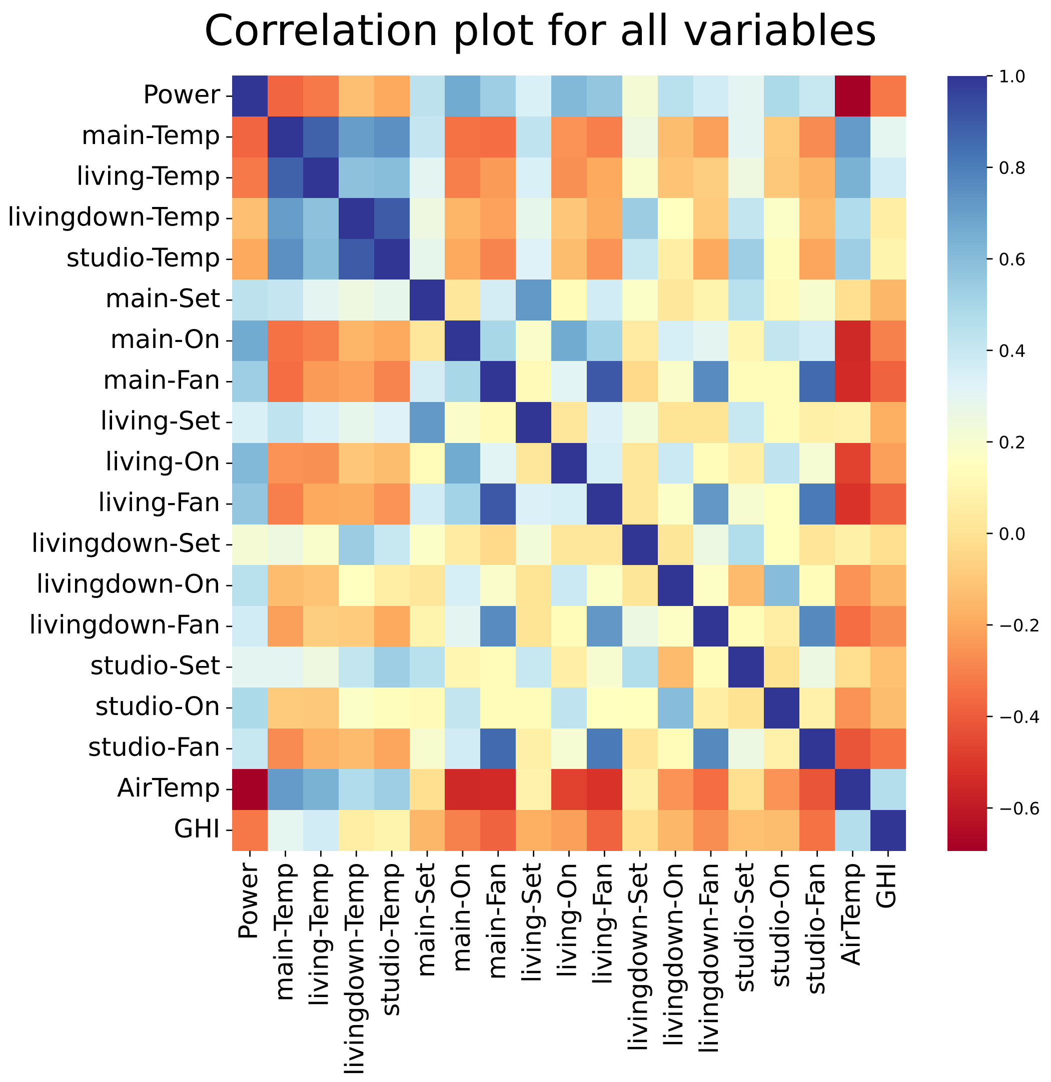

Figure 3.7: Correlations between the parameters.

4

Temperature Models

4.1 Modeling Temperature Dynamics

The temperature dynamics in the four rooms $R = \{main, living, livingdown, studio\}$ are to be modeled. This is done by creating time series multi-horizon forecasting models that forecast the room temperature M time steps into the future from time t where $t+1, ..., t+M$ are the forecast horizons. The MPC used on the system looks 12 hours into the future, and with a sampling frequency of 5 minutes, $M = 144$ forecast horizons are needed. The target variable $\boldsymbol{y}^r$ at time t is given by $\boldsymbol{y}^r = \left[y^r_{t+1}, ..., y^r_{t+M}\right]^\intercal$ for $r \in R$. The past N temperatures in all rooms are used as inputs to the forecasting models for each room. The size of N depends on the forecasting model. All past room temperatures are used to forecast the temperatures in each room. This is done to account for convection in the house. The inside temperatures do not follow any noticeable trends and lack seasonality but are heavily influenced by the heating of the heat pumps of all the rooms, outside temperature, and solar radiation as shown in Figure 3.7. Including these variables as future known independent variables is necessary to obtain a decent model. All independent variables used to create temperature dynamics models are explained in Table 4.1.

The heat pumps only affect the rooms' temperatures when they are turned on. To account for this in the data, the Set and Fan data is for each room multiplied with the respective boolean On setting. This prevents these data variables to affect the forecasted temperatures when the heat pumps are turned off.

The data set consists of time series data for all variables. Creating the training and test sets is done using the sliding window technique. The time series is split into overlapping sequences of length $N + M$ where each sequence begins on one later time step than the previous. With this approach, quite large training and test sets are created.

With the goal of developing linear models for temperature forecasting, direct and iterated approaches are detailed in the following sections in addition to the deep learning-based Temporal Fusion Transformer.

Table 4.1: Future independent variables for temperature forecasting.

Name	Explanation
main-Set	Target temperature in the main room
living-Set	Target temperature in the livingroom
livingdown-Set	Target temperature in the living room downstairs
studio-Set	Target temperature in the studio
main-On	Whether the heat pump in the main room is on or not
living-On	Whether the heat pump in the living room is on or not
livingdown-On	Whether the heat pump in the living room downstairs is on or not
studio-On	Whether the heat pump in the studio is on or not
main-Fan	Fan level in the main room
living-Fan	Fan level in the livingroom
livingdown-Fan	Fan level in the living room downstairs
studio-Fan	Fan level in the studio
AirTemp	Outside temperature from Solcast
GHI	Solar radiation from Solcast

4.2 Direct Linear Model

Linear regression methodology can be used to create linear forecasting models. Linear forecasting models have some similarities with the classical forecaster ARIMA. For this system, there is a lack of clear trends and seasonality in the temperature dynamics of the household and a need to include independent future variables. It is beneficial to create a linear forecasting model from scratch. A linear regression model that utilizes the past N values of a time series y up to time t with information about some future independent time series z can be used to forecast y at time $t+m$ by:

$$y_{t+m} = \sum_{i=0}^{N} w_{t-i}^{y} y_{t-i} + \sum_{i=0}^{m} w_{t+i}^{z} z_{t+i}, \tag{4.1}$$

or in vector form:

$$y_{t+m} = \begin{bmatrix} (\boldsymbol{w}^{y,m})^{\intercal} & (\boldsymbol{w}^{z,m})^{\intercal} \end{bmatrix} \begin{bmatrix} \boldsymbol{y}^{past} \\ \boldsymbol{z}^{future} \end{bmatrix} = \boldsymbol{w}^{\intercal}\boldsymbol{x}. \tag{4.2}$$

$\boldsymbol{w}$ denotes the model weights of size $(m+N)\times 1$ to be learned with $\boldsymbol{x}$ of the same shape. For simplicity, the sample indexing i is omitted in these and future equations. With $m=1$, the model makes one-step-ahead forecasts and can be used iteratively to produce forecasts M time steps into the future as detailed further in section 4.3. A direct model that forecasts values M horizons into the future can either be constructed using M direct models for $m \in [1, ..., M]$ or combining them into one matrix equation:

$$\boldsymbol{y} = \begin{bmatrix} y_{t+1} \\ ... \\ y_{t+M} \end{bmatrix} = \begin{bmatrix} (\boldsymbol{w}^{y,1})^{\intercal} & (\boldsymbol{w}^{z,1})^{\intercal} \\ ... & ... \\ (\boldsymbol{w}^{y,M})^{\intercal} & (\boldsymbol{w}^{z,M})^{\intercal} \end{bmatrix} \begin{bmatrix} \boldsymbol{y}^{past} \\ \boldsymbol{z}^{future} \end{bmatrix} = \boldsymbol{W}\boldsymbol{x}, \tag{4.3}$$

with $\boldsymbol{W}$ of shape $M \times (M+N)$ and $\boldsymbol{x}$ of shape $(M+N)\times 1$ for a single data sample.

With this as a basis, a linear deterministic direct forecasting model for the temperatures in room r, $\boldsymbol{y}^r$, M horizons into the future is given by

$$\boldsymbol{y}^r = \begin{bmatrix} y^r_{t+1} \\ y^r_{t+2} \\ \cdots \\ y^r_{t+M} \end{bmatrix} = \boldsymbol{W}^r \boldsymbol{x}, \tag{4.4}$$

where $\boldsymbol{x}$ consists of past target values of $\boldsymbol{y}^r$ for all rooms $r \in R$ and the independent variables $\boldsymbol{z}$ in Table 4.1:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{y}_{t-N} \\ \cdots \\ \boldsymbol{y}_t \\ \boldsymbol{z}_t \\ \cdots \\ \boldsymbol{z}_{t+M-1} \\ 1 \end{bmatrix}, \tag{4.5}$$

with $\boldsymbol{y_i} = \left[y_i^{main}, y_i^{living}, y_i^{livingdown}, y_i^{studio} \right]^T$ at time i. 1 is included to account for bias in the data. Because of inertia in the system, a change in the independent variables will not have an immediate effect on the room temperatures, and the independent variables are included from time step t to $t + m - 1$. With 14 independent variables and 4 rooms, $\boldsymbol{W}^r$ is of size $M \times (14 \cdot M + 4 \cdot N + 1)$ and consists of the model parameters.

There are four rooms, and the number of forecast horizons M is set to 144. The past three hours of room temperatures are used in this model, resulting in a look-back window of size $N = 36$. There are 14 future independent variable time series that are used for forecasting. These time series should be limited such that only their values prior to the forecast horizon m affect the forecast at said horizon. Equation (4.1) presents a causal forecaster - a property that must be fulfilled. This is not the case for the non-causal system model presented in (4.4) where all independent variables affect all forecasts. The parameter weights that affect future forecasts must be constrained by setting them to zero to ensure causality in the system.

Finding the optimal values of $\boldsymbol{W}^r$ in the non-causal (4.4) can be done using the ordinary least squares method as detailed in section 2.2. The sum of squared errors is to be minimized for the forecasting model with respect to $\boldsymbol{W}^r$. With n data samples, the optimal solution is given by

$$\hat{\boldsymbol{W}}^r = \underset{\boldsymbol{W}^r}{\arg\min} \frac{1}{n} \mid\mid \boldsymbol{Y}^r - \boldsymbol{W}^r \boldsymbol{X} \mid\mid^2 = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{Y}^r, \tag{4.6}$$

where $\boldsymbol{Y}^r$ containing the future target time series for room r is of shape $M \times n$ and $\boldsymbol{X}$ is of shape $(14 \cdot M + 4 \cdot N + 1) \times n$ and contains the past target time series and independent variable time series. The assumptions of linearity are not entirely met in this problem. There is not a strictly linear relationship between the independent and dependent variables, and there is some correlation between the independent variables. However, OLS may still produce reasonable results. The aforementioned causality constraint increases the problem's complexity. OLS is not capable of directly adhering to the constraints, and some other methodology must be used. Two possible approaches are:

1. Solving OLS problems for each forecast horizon. In this approach, only the independent variables prior to the forecasting horizon are included.

2. Using a projected gradient descent method as presented in 2.3 where a causality matrix S is used to mask the gradient at each iteration to ensure causality. This method may use the non-causal OLS solution for (4.4) for initializing the weights.

Both these approaches require substantially more processing resources than only using OLS, but the gradient descent method is more efficient than the other approach on large data sets. It is therefore chosen going forward. When training the linear deterministic models using gradient descent, the MSE is used as a loss function. The same method is used to produce stochastic forecasts by utilizing MAD instead. Forecasts at different quantiles are produced by training several models at the specified quantiles. Training both linear direct deterministic and stochastic models for the temperature dynamics in each room $r \in R$ is done by performing the following steps:

1. The model weights are initialized using the OLS solution of (4.4) masked by a causality matrix S. S ensures that independent variables for a time t do not affect forecasts at earlier horizons.

2. Projected gradient descent is performed where updates to the model weights are masked by the causality matrix to find a causal minimum of the MSE for deterministic models and the MAD for stochastic models.

For the stochastic models, the steps are repeated for the quantiles $q \in \{0.1, 0.5, 0.9\}$ resulting in three models for each of the four rooms. The causal $\boldsymbol{W}$ consists of 167 040 parameters per model with the chosen specifications. The forecast at forecast horizon $m = 1$ is affected by 159 parameters, while the forecast at $m = 144$ uses 2161 model parameters. Training one model on fifty weeks of data on a decent computer takes about 20 minutes. With only one week of data, that reduces to a couple of minutes. With 167 040 active parameters and a total size of 311 184 parameters including all zero-valued parameters, one could assume that using this model in an MPC scheme would slow it down. By enforcing causality, $\boldsymbol{W}$ has a strong structure that the MPC can exploit. It is similar to structures used in condensed MPCs, and this can be exploited to increase the efficiency of solving the MPC optimization problem.

4.3 Iterated One-Step-Ahead Linear Forecasting Model

As detailed in the previous section, a one-step-ahead forecaster for the temperature in room r is given by:

$$y^r_{t+1} = (\boldsymbol{w}^r)^\intercal \boldsymbol{x}. \tag{4.7}$$

with

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{y}_{t-N} \\ \dots \\ \boldsymbol{y}_t \\ \boldsymbol{z}_t \\ 1 \end{bmatrix}, \tag{4.8}$$

As for the direct model, $\boldsymbol{y}_k$ is the temperatures in the four rooms at time k, and $\boldsymbol{z}_t$ the values of the independent variable detailed in Table 4.1 at time t. A forecast $M = 144$ time steps into the future is performed by iteratively making one-step-ahead forecasts M times with a rolling window approach. The window of past temperatures is moved along the forecast time where the past N measured and previously forecasted temperatures are used as input to the models. The independent variables at time $t + m$ are used for the forecast for time $t + m + 1$. The size of the lookback window is set to $N = 36$, the same as for the direct model. $(\boldsymbol{w}^r)^\intercal$ is of size $(4 \cdot N + 14 + 1) \times 1$ resulting in a total of 159 parameters. This is significantly less than the whole direct linear forecasting model. This model is inherently causal, which means OLS can be used to find the $\boldsymbol{w}$ that minimizes the sum of squared errors for the model for room r:

$$\hat{\boldsymbol{w}}^r = \arg\min_{\boldsymbol{w}^r} \frac{1}{n} \mid\mid \boldsymbol{y}^r - \boldsymbol{w}^r \boldsymbol{X} \mid\mid^2 = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}^r, \tag{4.9}$$

for n data samples and where $\boldsymbol{y}^r$ contains the future target time series is of shape $1 \times n$ and $\boldsymbol{X}$ is of shape $(4 \cdot N + 14 + 1) \times n$ containing the past target time series and the 14 independent variables.

Deterministic one-step-ahead models are trained for each room using this approach.

4.4 Temporal Fusion Transformer

To compare the performance of linear forecasting models to state-of-the-art deep learning techniques for indoor temperature forecasting, a Temporal Fusion Transformer model is trained on the data. The TFT is a direct model that combines past target values, and past and future time-dependent independent variables in addition to static independent variables that characterize different time series. It is also a stochastic model and may use a quantile loss function to forecast values at different percentiles. [1] presents all aspects of the model, and Appendix A contains an extract from my Project Thesis on the TFT architecture. An implementation of the TFT by Darts [32] is used in this thesis.

As with the linear models, we want the TFT to forecast temperatures 12 hours, or 144 time steps, into the future. The TFT support multiple output time series and only one model is needed to provide a stochastic forecast for all four rooms. The trained model forecasts the temperatures in all four rooms for quantiles $q = \{0.1, 0.5, 0.9\}$. The TFT learns and utilizes patterns in past target variables, so the number of past time steps should be considerably larger than the number of forecast horizons. The room temperatures in the past 24 hours, or 288 time steps, are used for forecasting. The independent variables in 4.1 are also provided to the model as future known variables. Past and static independent variables are not used in this configuration. Masking mechanisms inside the model preserve causality. All independent variables are scaled using min-max normalization to be in the range of $[0, 1]$. Most of the variables do not have a gaussian distribution, and this method is thus better than the standardization technique. The output of the TFT is calculated using a sigmoid activation function which has a range of zero to one. The dependent variables are also scaled using min-max normalization to be in that same range.

Other important tunable model parameters include the hidden state size (capacity of the model where a larger hidden size allows the model to capture more complex patterns in the data), the number of LSTM layers, and the number of attention heads (for learning global dependencies). These are set to the recommended default values 64, 1, and 4 respectively. More details on how these parameters affect the model is provided in Appendix A.

The TFT is trained by minimizing the quantile loss summed over all quantile outputs using gradient descent methods. With the specified configuration, the TFT model consists of 267 000 trainable parameters. This is almost twice the number of parameters in the linear direct model, but due to the complex model architecture, training the model for a single epoch using a Central Processing Unit (CPU) takes up to 2 hours on 50 weeks of data. Training the model with a sufficient number of epochs takes days while the linear model is trained in about 20 minutes with the same amount of data. This makes tuning the TFT a time-consuming task where a small increase in performance may not justify using weeks of fine-tuning the model in this context. Training the TFT on one week of data takes only approximately an hour. The use of Graphics Processing Units (GPUs) was not possible during the work on this Theis, but utilizing them instead of CPUs would decrease the training times. GPUs consist of many small simple processors that make parallel computations possible and are well-suited for handling the large data sets and computations required in deep learning algorithms.

4.5 Model Evaluations

This section presents the performance of the created models on test data. Stochastic and deterministic direct linear forecasting models, and deterministic iterated one-step-ahead forecasting models are created and trained on training data of length one to fifty weeks. In addition, a Temporal Fusion Transformer model is created and trained on one and fifty weeks of data. The stochastic models are trained to output forecasts for quantiles $q \in \{0.1, 0.5, 0.9\}$

4.5.1 Direct Linear Models

Figure 4.1 presents the mean absolute errors for the deterministic and stochastic direct linear models on the test data of increasing length. The evolution of errors for the temperature differs, but generally, it decreases for a longer training set. The error for the deterministic and median of the stochastic models are also quite similar. This is confirmed in the more detailed Figure 4.2 which also includes the error for each forecast horizon for the models trained on a selection of training sets.

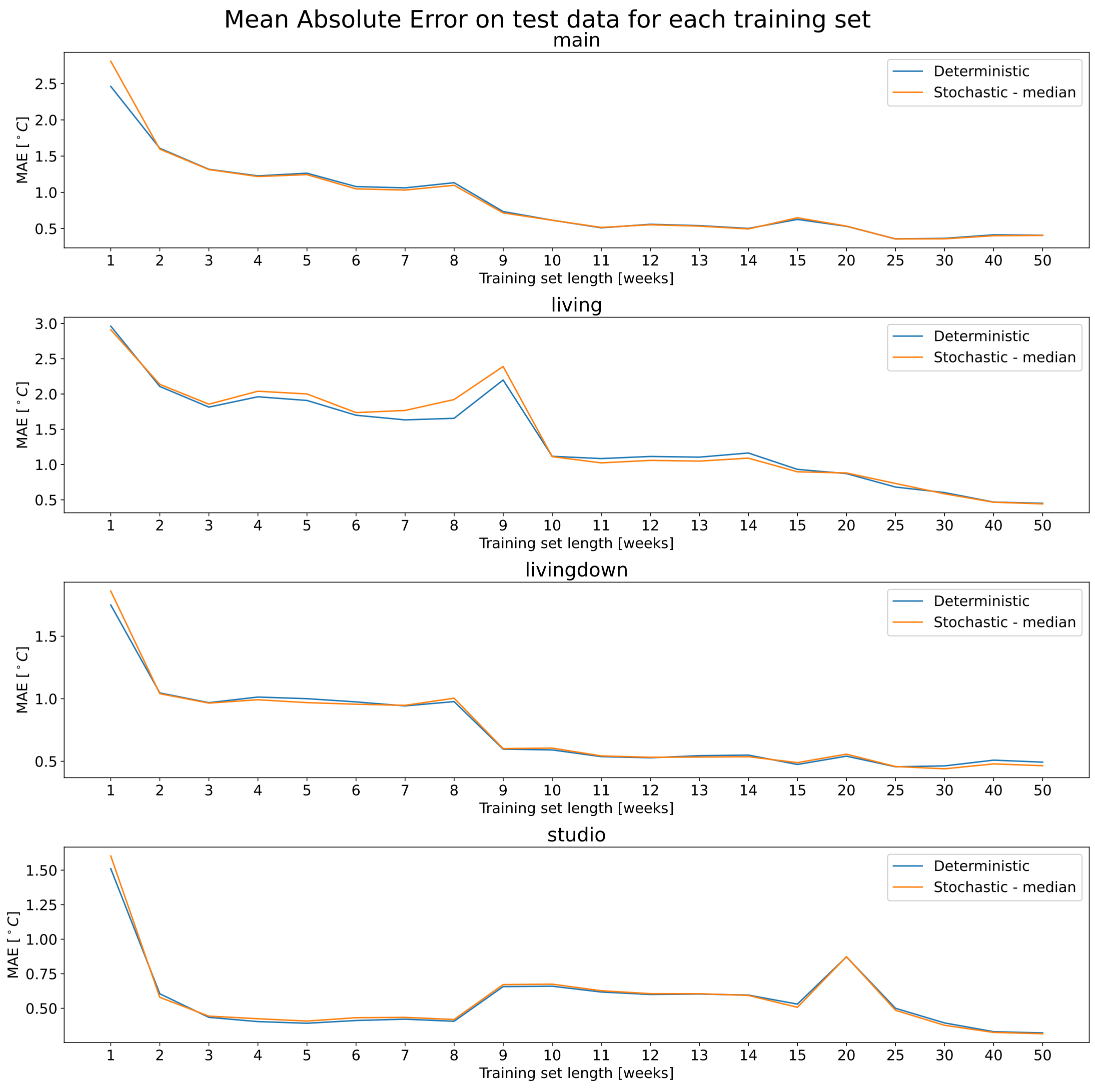

Figure 4.1: Mean absolute errors on test data for the linear direct models.

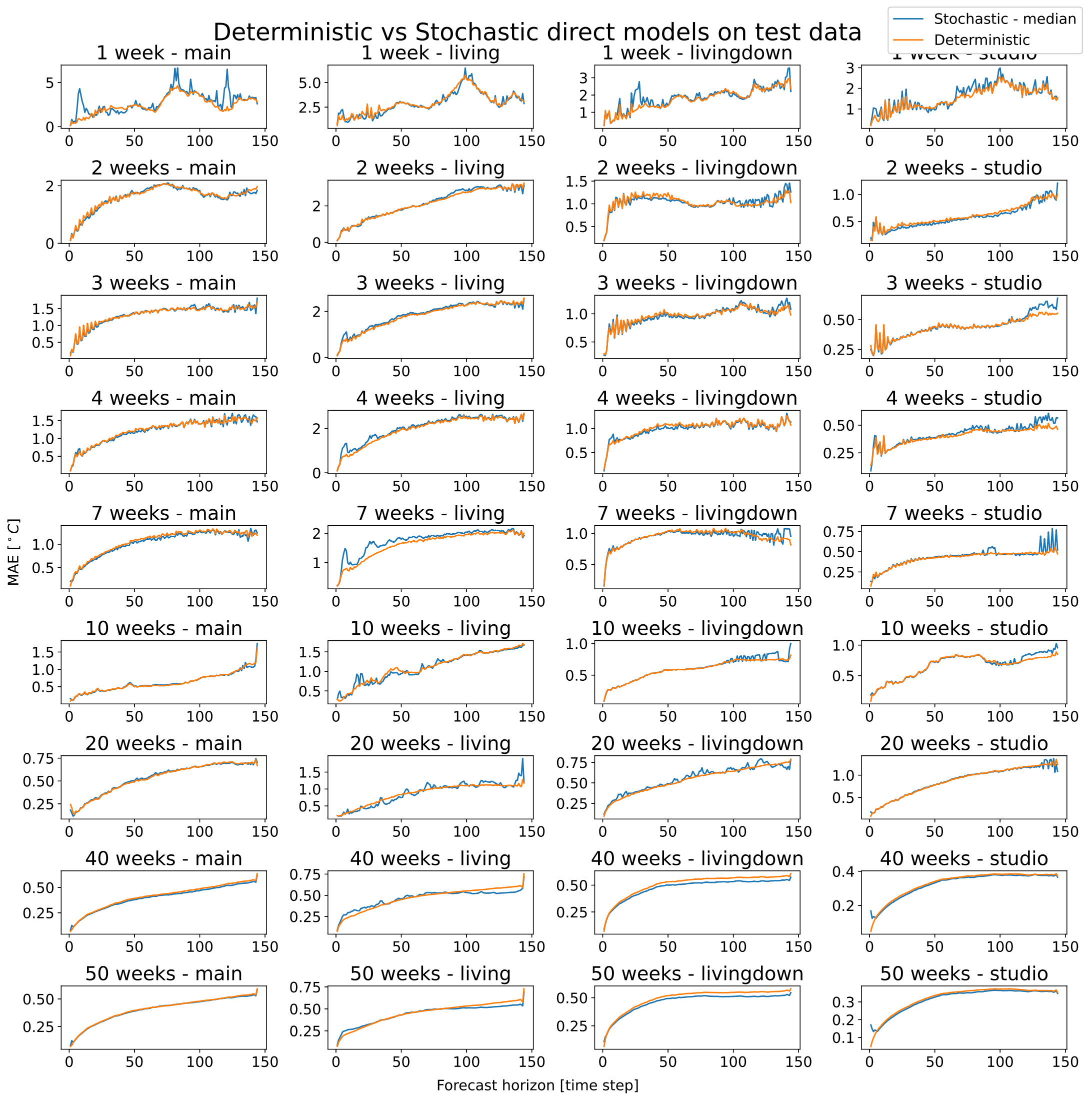

Figure 4.2: Mean absolute errors on test data for all forecast horizons for linear deterministic and stochastic models trained on a selection of training sets.

The performance of the deterministic model is slightly better than the stochastic for little training data, but for longer training sets, the difference is negligible. Figures 4.3 and 4.4 present the residuals for the deterministic and median of the stochastic models for the temperature in each room. The models have quite similar residuals with a slight positive bias for the later forecast horizons. This means the models tend to forecast too high temperatures on the test data. The residuals for the lower forecast horizons are quite better than the later ones, but with sufficient training data, most errors are within $1°C$.

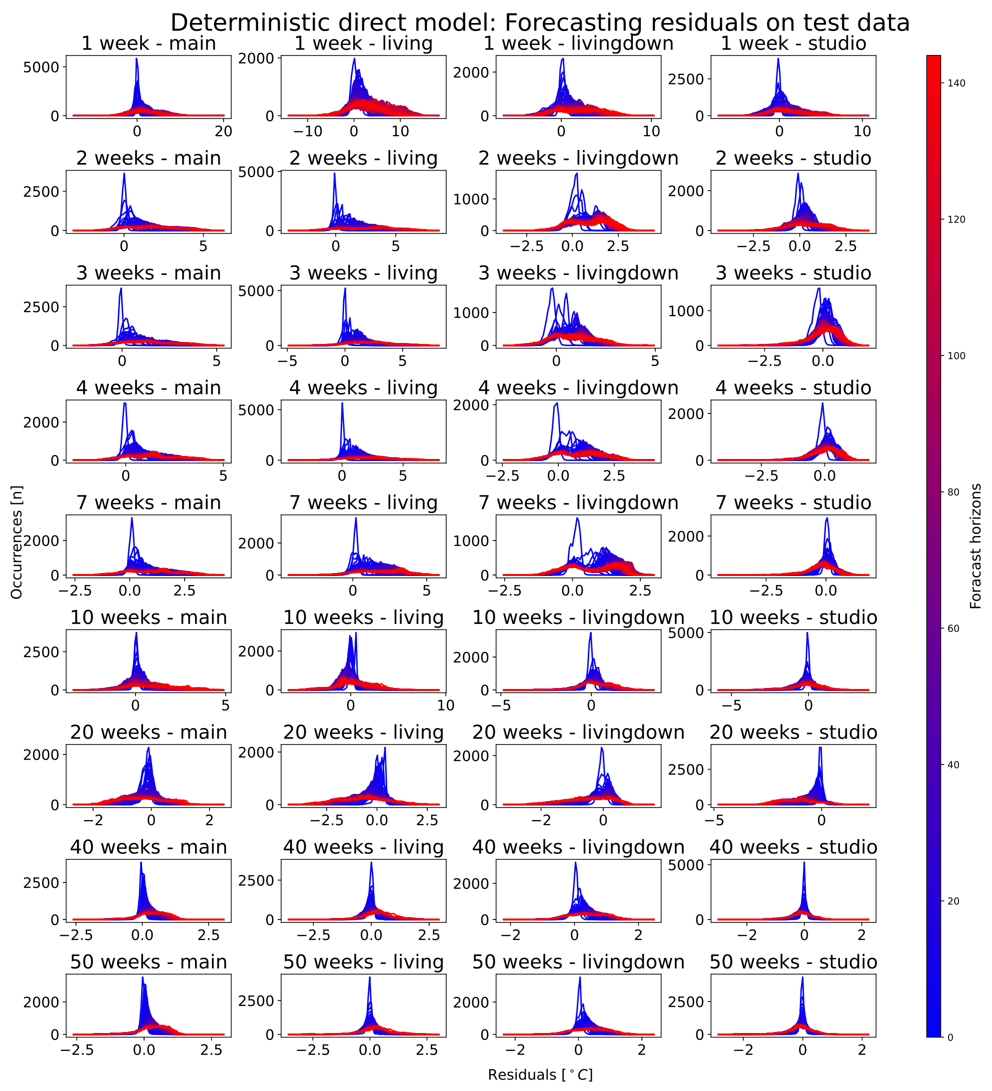

Figure 4.3: Residuals for the deterministic linear model on the test data for a selection of trained models.

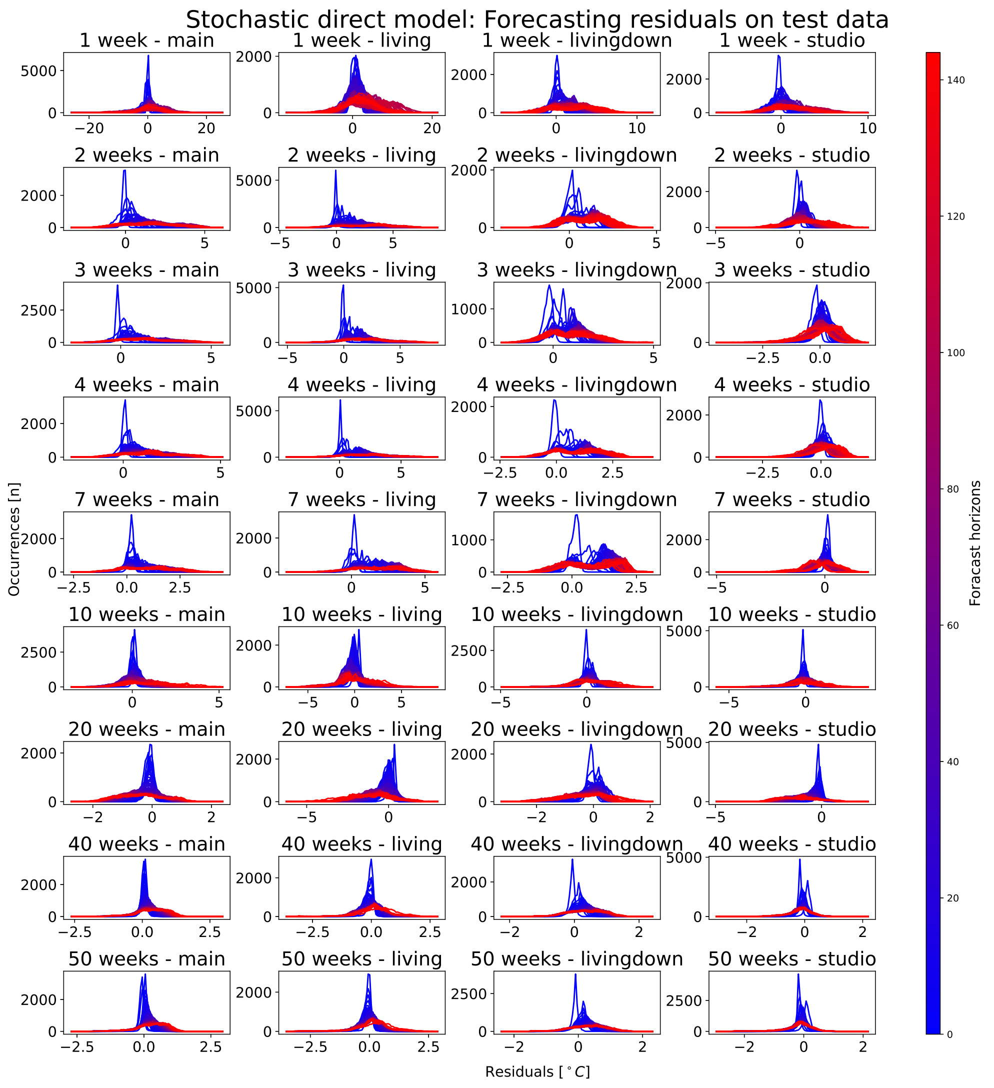

Figure 4.4: Residuals for the median of the stochastic linear model on the test data for a selection of trained models.

To study the complete performance of the stochastic models, the upper and lower percentiles have to be accounted for. Figure 4.5 presents the expected error for each quantile on the test set. These errors are a measure of the uncertainty of the model predictions. As expected, the magnitude of the quantile errors decreases for increasing training set lengths. However, some of the models are heavily biased with the 0.1 quantile forecast above $0°C$, meaning that they fail to accurately forecast the uncertainty of the data. For the larger training sets, the quantile errors stabilize with slightly increasing uncertainty for increasing forecast horizons. The expected error of the median forecast is positively biased for most of the trained models. This bias increased with the forecast horizons. For the 20 week-model, the forecasts are negatively biased. The model trained on fifty weeks of data has the smallest expected errors with a magnitude of less than $0.5°C$ for all forecast horizons.

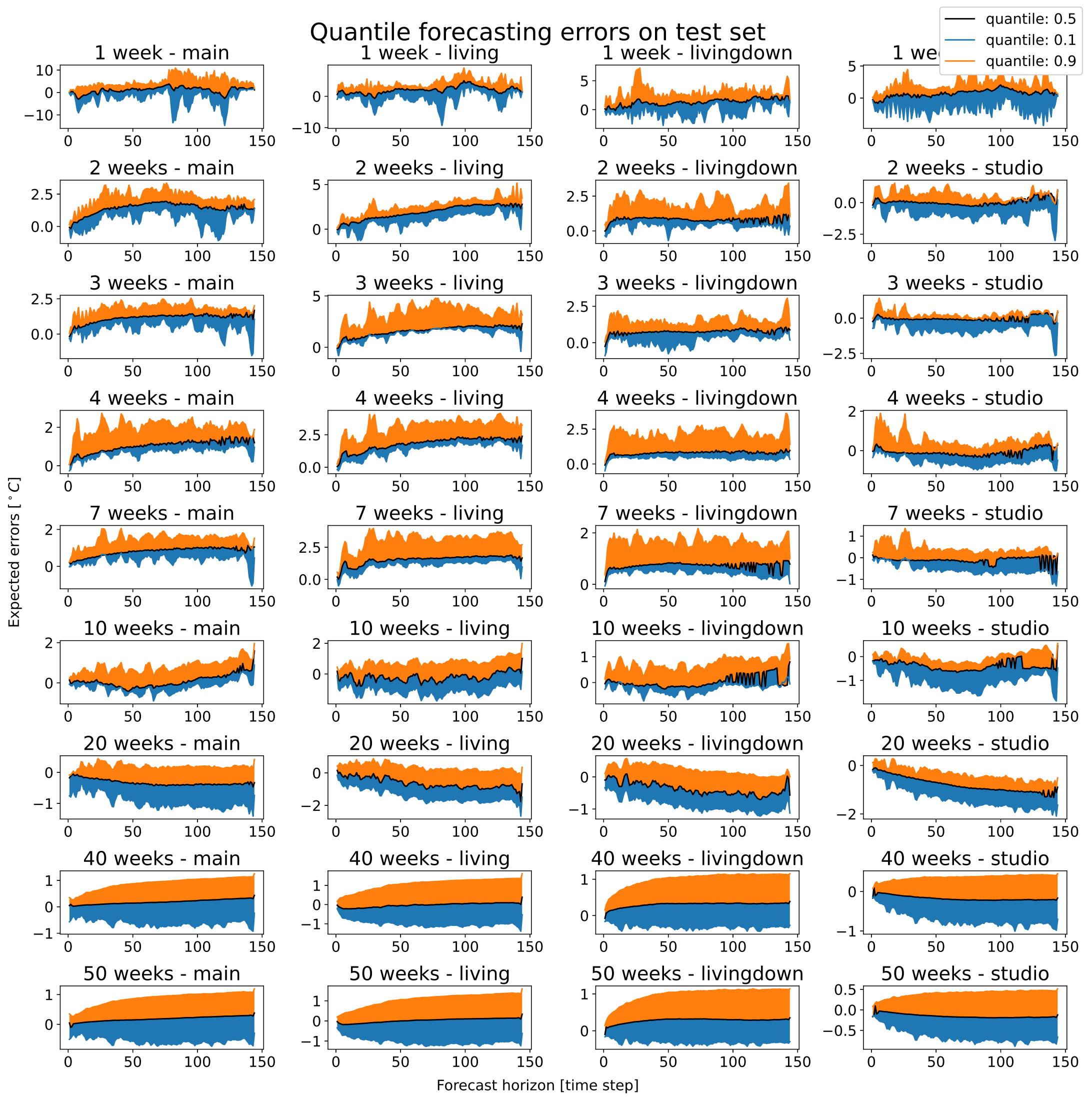

Figure 4.5: Expected errors for each quantile at each forecast horizon for a selection of trained models.

4.5.2 Iterated One-Step-Ahead forecaster model

In Figure 4.6, the mean absolute errors for all forecast horizons are shown for the deterministic direct model and the iterated one-step-ahead forecaster. While the initial one-step-ahead forecast some times are better than the direct forecaster, the iterated model performs significantly worse than the direct model. For some of the training sets, the error diverges and is several times larger than the direct version and iterated models trained on shorter data sets. This is illustrated in Figure 4.7. The residuals for the iterated one-step-ahead models are presented in Figure 4.8. For most of the models trained on ten weeks of data or more, there is little bias in the forecasts, but there are some large forecast errors. The models trained on fewer data samples have both large positive and negative residuals.

All this shows that for this forecasting problem, a linear one-step-ahead forecaster does not manage to capture the temperature dynamics sufficiently with the available data, and a more robust direct linear model is needed.

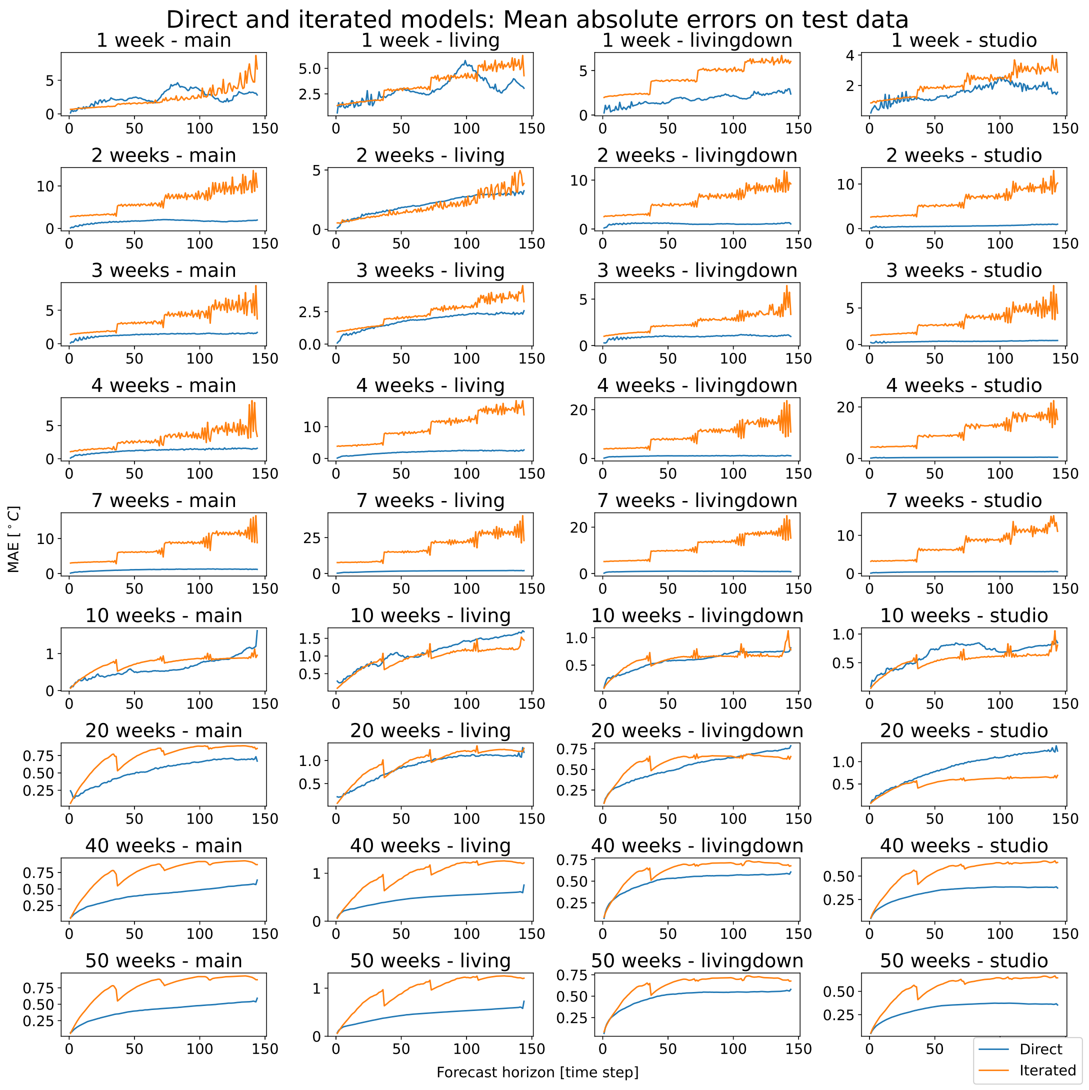

Figure 4.6: Mean absolute errors on test data for all forecast horizons using the direct and iterated linear deterministic models trained on a selection of training sets.

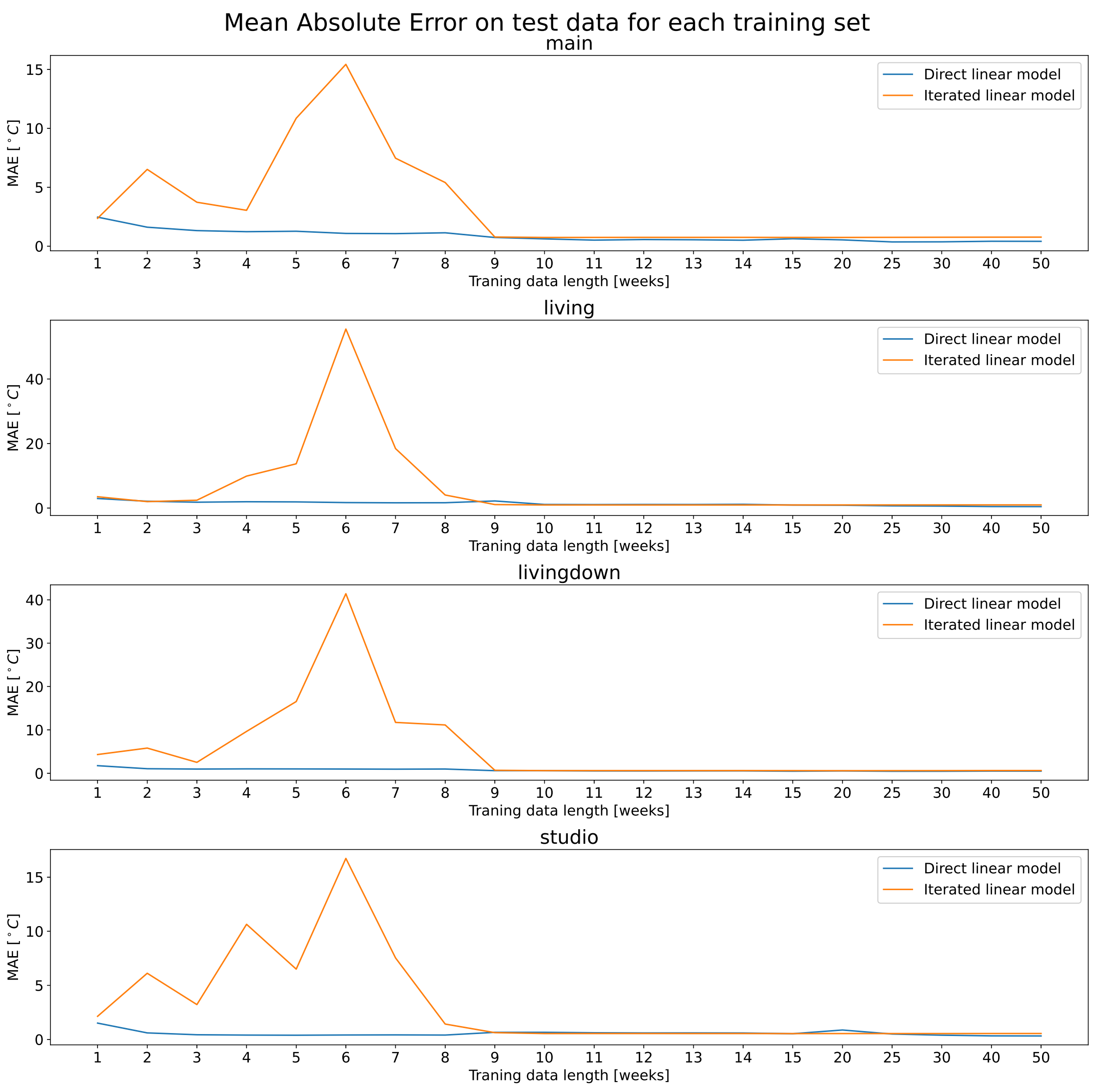

Figure 4.7: Mean absolute errors on test data for the linear direct and iterated models.

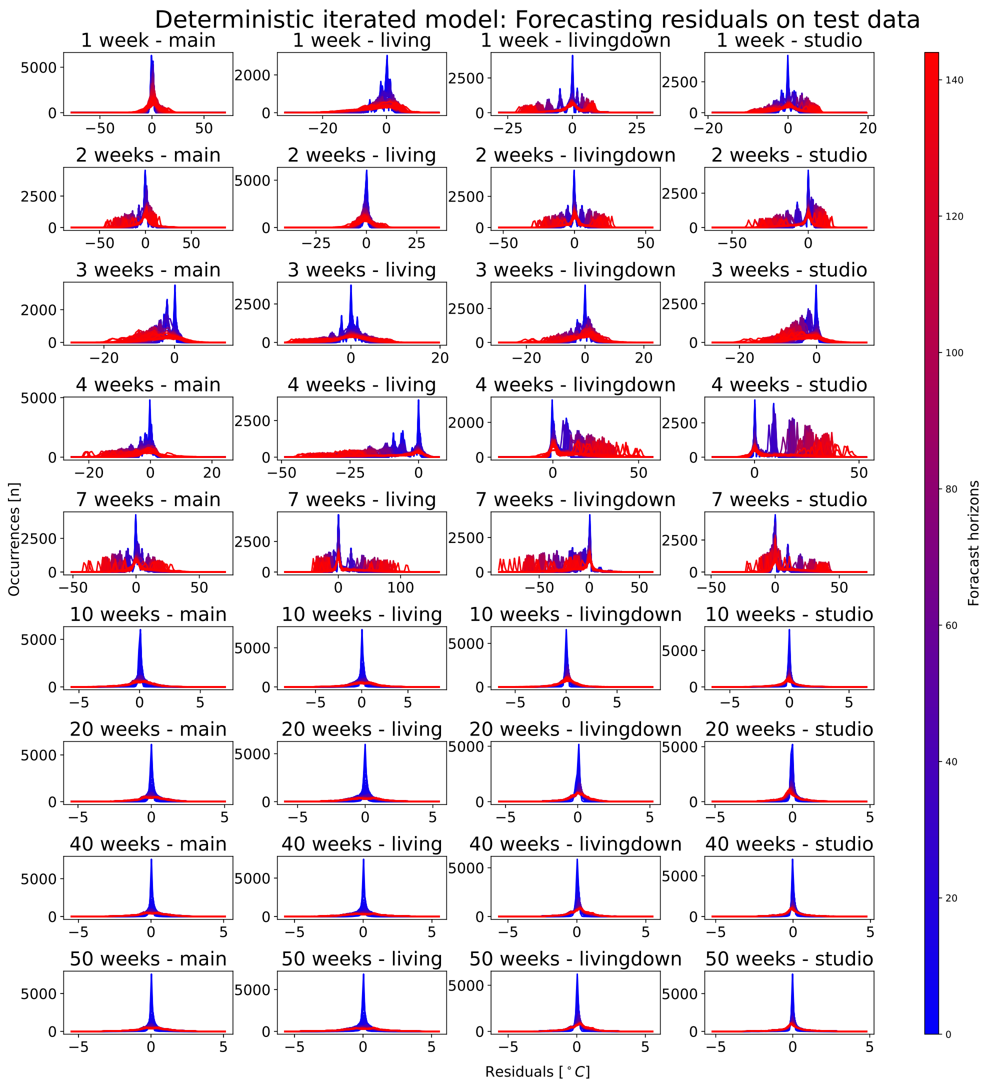

Figure 4.8: Residuals for the median of the iterated linear model on the test data for a selection of trained models.

4.5.3 Temporal Fusion Transformer

In this thesis, two models using the TFT were created. One that is trained on one week of data, and one that is trained on 50 weeks of data. Figure 4.9 presents the mean absolute error for all forecast horizons for the TFT model trained on one week of data. It is compared with the linear direct models trained on two, three, four, and ten weeks of data. As the direct linear models trained on one week of data has quite poor performance, it is not shown here. For the early forecast horizons, all the linear models outperform the TFT. For the later forecast horizons, the linear direct model needs to be trained on ten weeks of data for most of the rooms to outperform the TFT trained on one week of data. In contrast to the linear models, the mean absolute errors of the TFT are quite stable for all forecast horizons, with little oscillations.

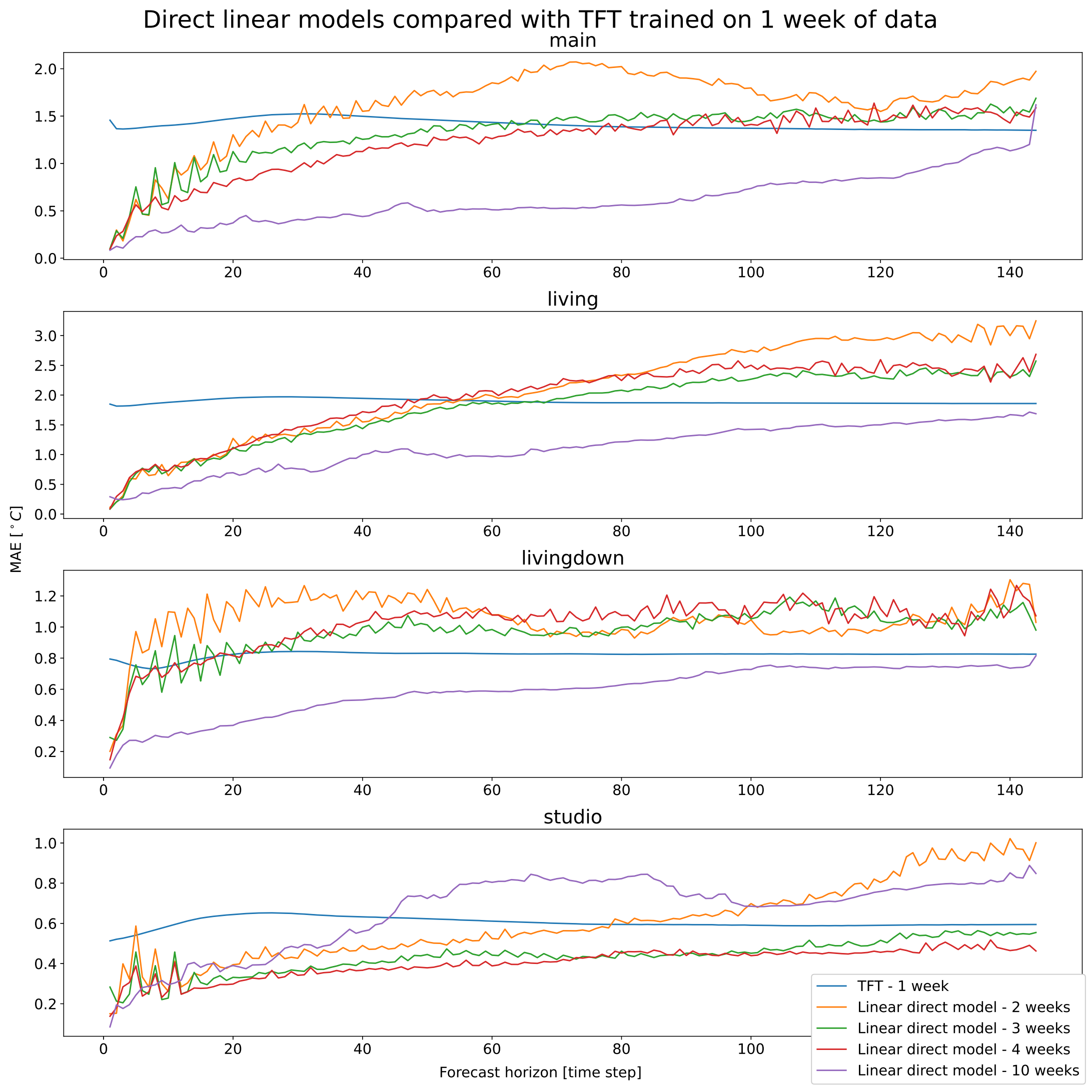

Figure 4.9: MAE for all forecast horizons for TFT model trained on one week of data compared to the direct linear model.

The mean absolute errors for all room temperatures for all forecast horizons for the TFT model trained on 50 weeks of data are presented in Figure 4.10 compared with deterministic direct linear models trained on 30, 40, and 50 weeks of data. The relative performance of the TFT and linear models differ for the four rooms. The linear models all outperform the TFT in the early forecast horizons - as for the TFT trained on one week. The linear models have a notable increase in the mean absolute errors for the later forecast horizons while the TFT provides a more similar error for all forecast horizons. However, with all errors differing with maximum $0.5°C$, the difference is indistinguishable.

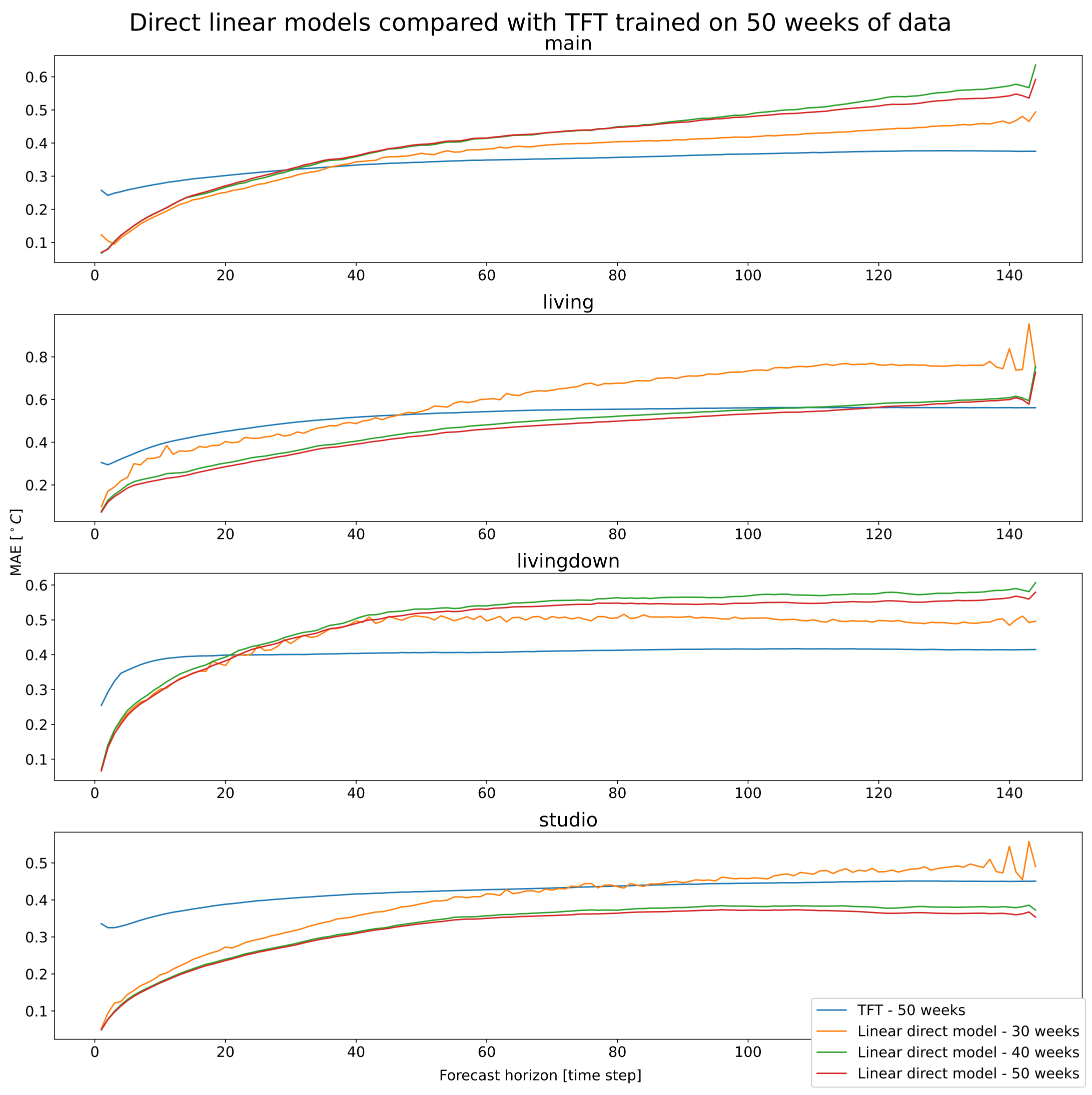

Figure 4.10: MAE for all forecast horizons for TFT model trained on one week of data compared to direct linear model

Figure 4.11 presents the residuals for both TFT models. These plots confirm that the errors for all forecast horizons are quite similar. While the errors for the TFT trained on one week of data are biased, there is little bias for the TFT trained on fifty weeks of data. Figure 4.12 shows the quantile mean errors for the TFT models. While the stochasticity of the linear direct model manages to capture the increasing uncertainty of the forecasts for the later forecast horizons, The TFT is equally unsure of forecasts one step ahead and 144 steps ahead. The expected errors for the one-week TFT are quite large for two of the rooms. For the TFT trained on 50 weeks of data, the expected error is smaller in magnitude but not much smaller than the direct linear model trained on the same amount of data.

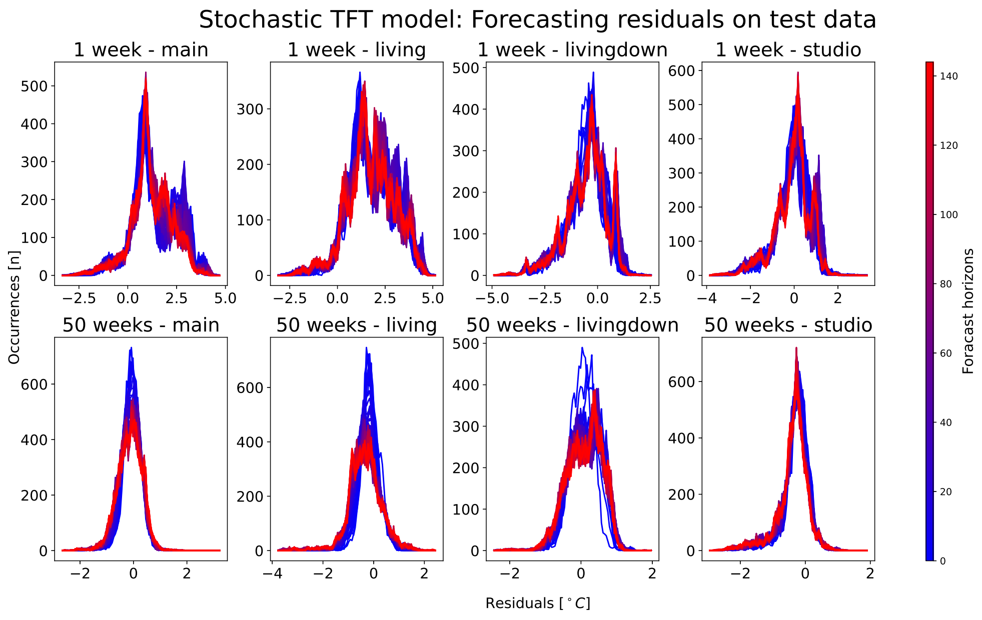

Figure 4.11: Distribution of residuals for the TFT models.

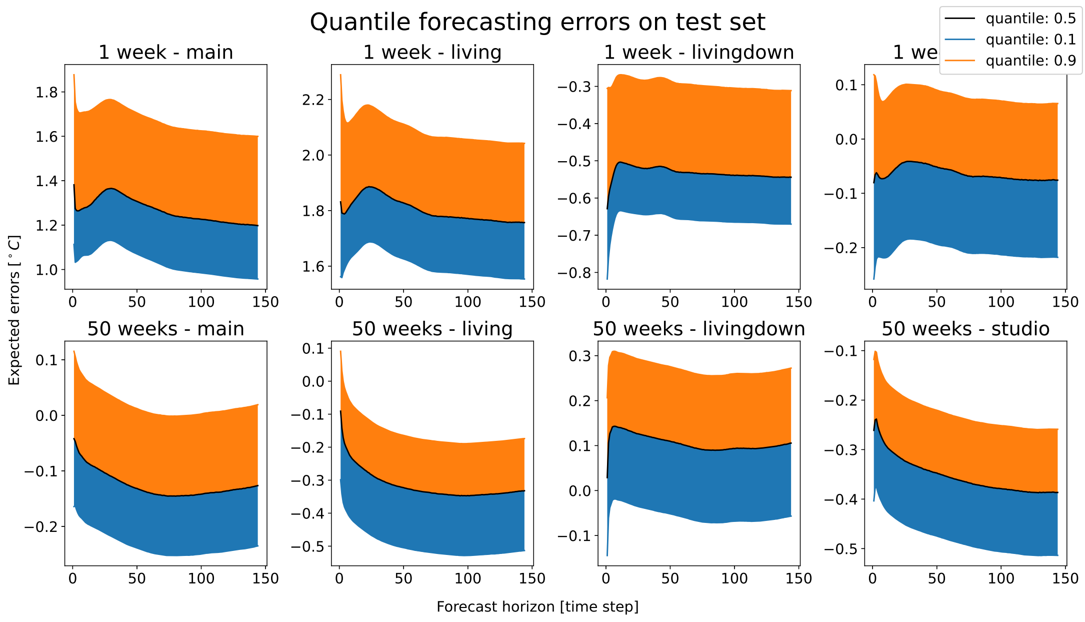

Figure 4.12: Quantile errors for the Temporal Fusion Transformer models.

4.6 Analysis of the Direct Linear Model

As described in section 4.2, ordinary least squares were used to find the initial model parameters for the direct linear forecasting model. This solution is non-causal, and the causal model is found using masking combined with a gradient descent algorithm. Figure 4.13 presents the absolute forecasting errors for all three steps of the method on the training sets. Comparing the masked and unmasked least squares-based models, the effect of masking is large when little data is used, but for the longer data sets, the effect is smaller. Consequently, the following gradient descent algorithm has poor initial values for the smaller data sets, but quite good initial values for the longer ones. The causal optimal solution achieved by the projected gradient descent method manages to close the gap to the non-causal least squares solution for most of the training sets. For the larger training sets, the difference between the initial non-causal OLS solution and the final causal model is indistinguishable. This is as expected since the system we model is causal.

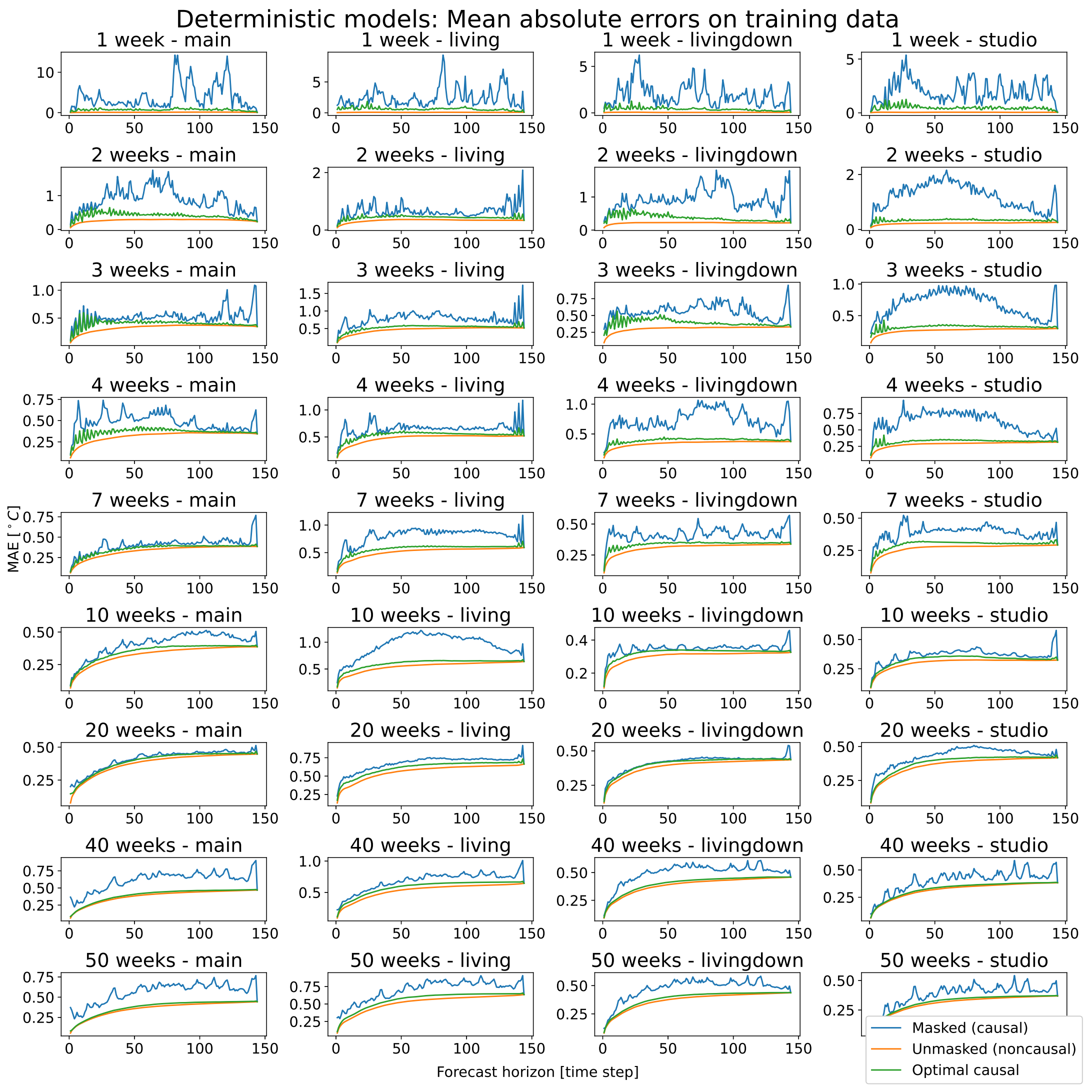

Figure 4.13: Mean absolute errors on test data for all forecast horizons using the least squares solution, masked least squares solution, and the final model achieved using gradient descent with the masked least squares solution as initial values.

The past 3 hours (or 36 time steps) of room temperature data are used in the direct linear forecasting model for all four rooms. Figure 4.14 presents the mean weights over all forecast horizons for each lagged data point for the deterministic model trained on 50 weeks of data. As expected, the temperatures closest in time affect the forecast more, with the temperature in each room having the biggest effect on each respective room temperature forecast. The model also captures the temperature dynamics between each room. For example, the temperature in $living$ has a greater impact on the forecasted temperature in $main$ than in the other two rooms. The same is true for the two rooms downstairs. The model weights converge to zero quite fast for the lags greater than 5 time steps. The true temperature dynamics are not that quick. For the model weights to accurately model the true dynamics, the weights should perhaps be regularized to prevent steep changes in the absolute magnitude. The mean weight values for each room temperature over all lags for all forecast horizons are presented in Figure 4.15. These plots show that for the first forecast horizons, the forecasted temperatures are mainly affected by the past temperatures of each respective room. For the later forecast horizons, the past temperatures in the other rooms slowly become more relevant, and their weights increases in magnitude. This demonstrates the temperature dynamics between each room.

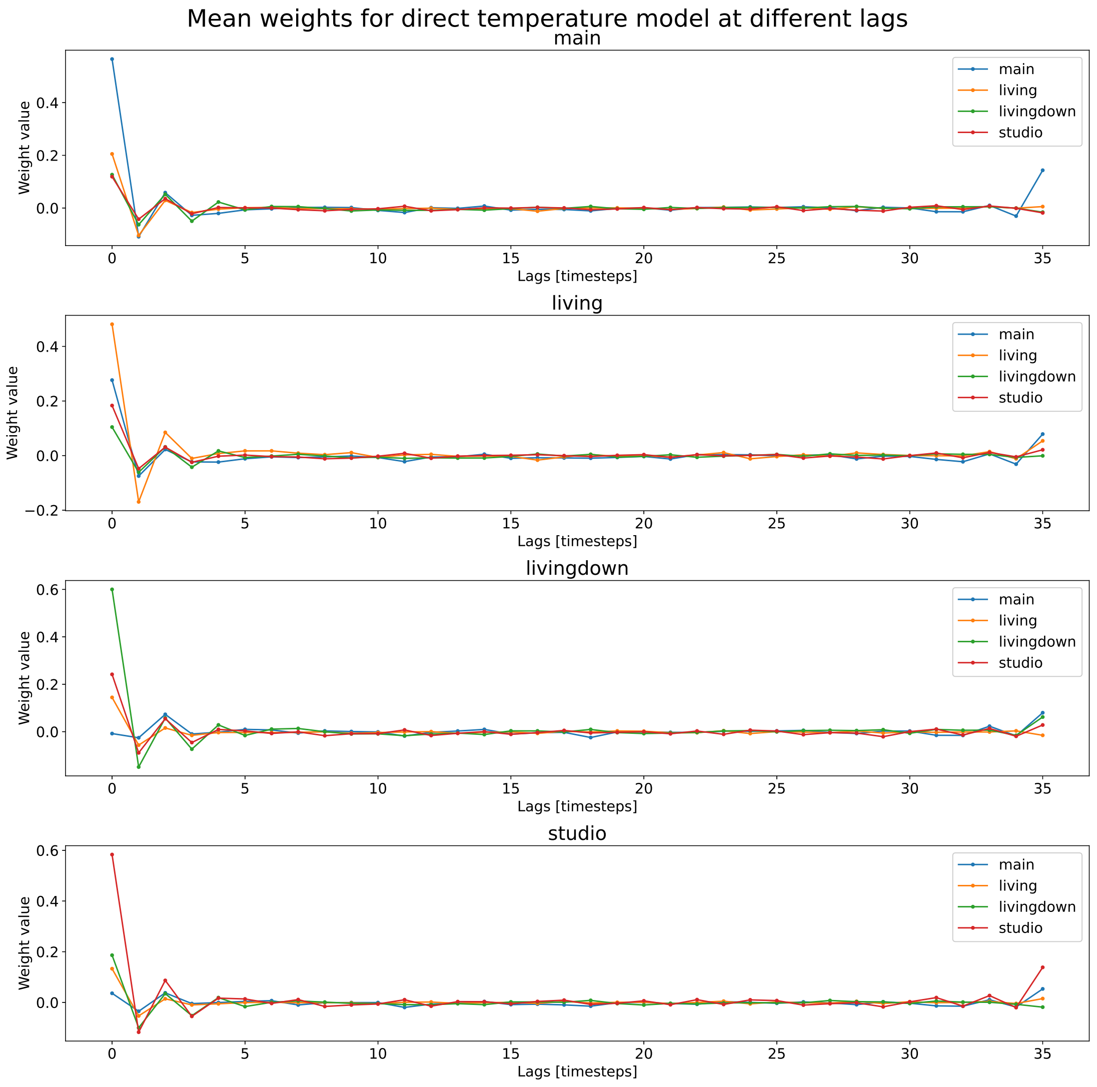

Figure 4.14: Mean weight values for the lagged temperature values for each past time step in the deterministic direct linear model.

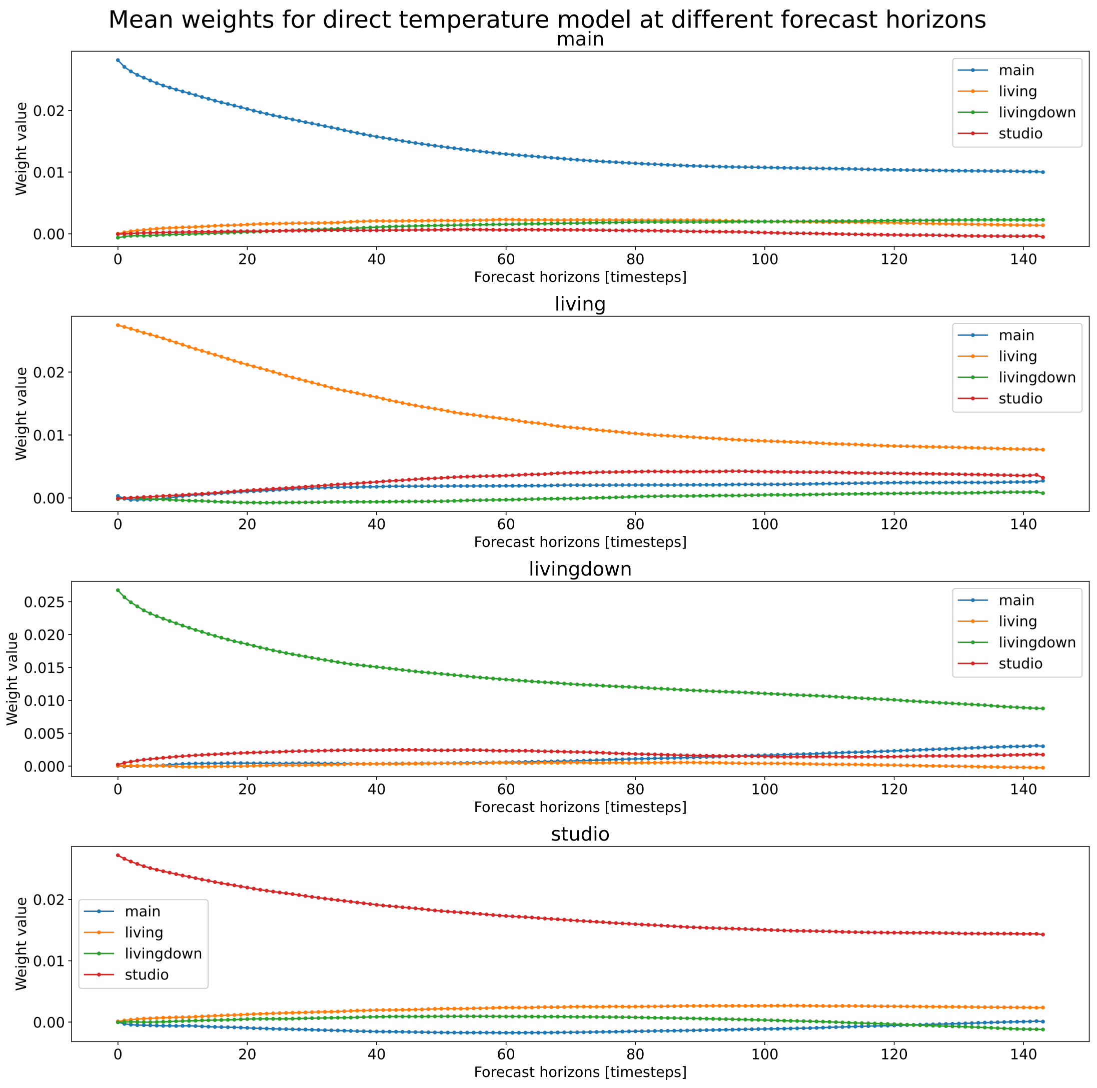

Figure 4.15: Mean weight values for the lagged temperature values for each forecast horizon in the deterministic direct linear model.

All the independent variables have different distributions. This makes it difficult to assess their relative contribution to the forecasted temperatures in each room. With one forecasting model for the temperature in each room, parameter importance may only be compared for each model where the parameters have the same distribution. This means that the relative contribution from outside temperature and radiation for each room temperature forecast cannot be assessed. The individual heat pump settings in each room have the same distribution. Their contribution to the forecasted temperature in each room can therefore be analyzed. Figures 4.16, 4.17, and 4.18 present the model weight magnitudes for *Set*, *Fan*, and *On* for the deterministic direct linear model trained on 50 weeks of data respectively. Each row of plots presents how the given setting for each indoor heat pump unit affects the temperature forecast for a certain room. These plots show that the model captures which room the indoor heat pump units are situated in. The plots also show how the settings in all the rooms affect the forecasted temperature in each room. *main* and *living*, and *livingdown* and *studio* affect each other the most. Settings closest in time to the forecast have the highest importance for the forecasts, but for *Set*, past values are also given some precedence.

Note the vertical lines in the plots for *Set* and *Fan*. These lines show that parameter values at certain time steps are equally important for forecasts at several horizons. In true temperature dynamics, the importance of a variable decreases the further into the past it is sampled. To ensure that the models adhere to this physical property, the model weights could be regularized in time. The blue lines in the *On*-plots mean that the model forecasts lower temperatures when the heat pumps are on than when it is off. This is the opposite as expected since it normally gets colder inside when the heat pumps are off for most of the year in Norway. One explanation could be that this phenomenon results from periods when it is warmer than the *Set* value. In these periods, the heat pumps will be off, but the temperature will be higher than normal. The heat pumps are only turned on again by the control scheme when the temperature is lower and heating is needed.

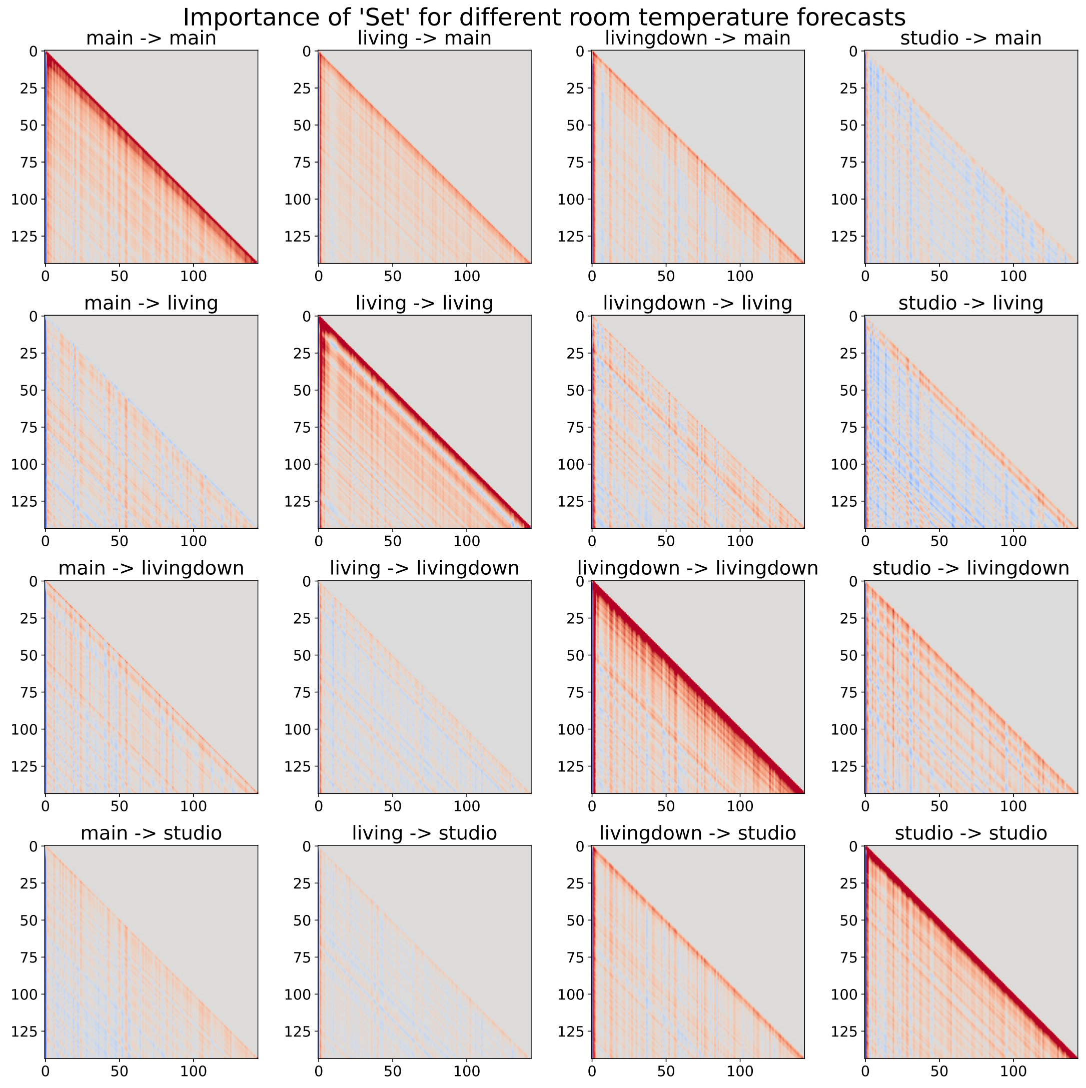

Figure 4.16: Model weight magnitudes for the 'Set'-setting for the heat pump units in each room for all forecast horizons (along the y-axis) at all future time steps (along the x-axis). The colors red and blue signify positive and negative values respectively. Zero-valued weights are represented by the color gray. The model is the deterministic direct linear model trained on 50 weeks of data.

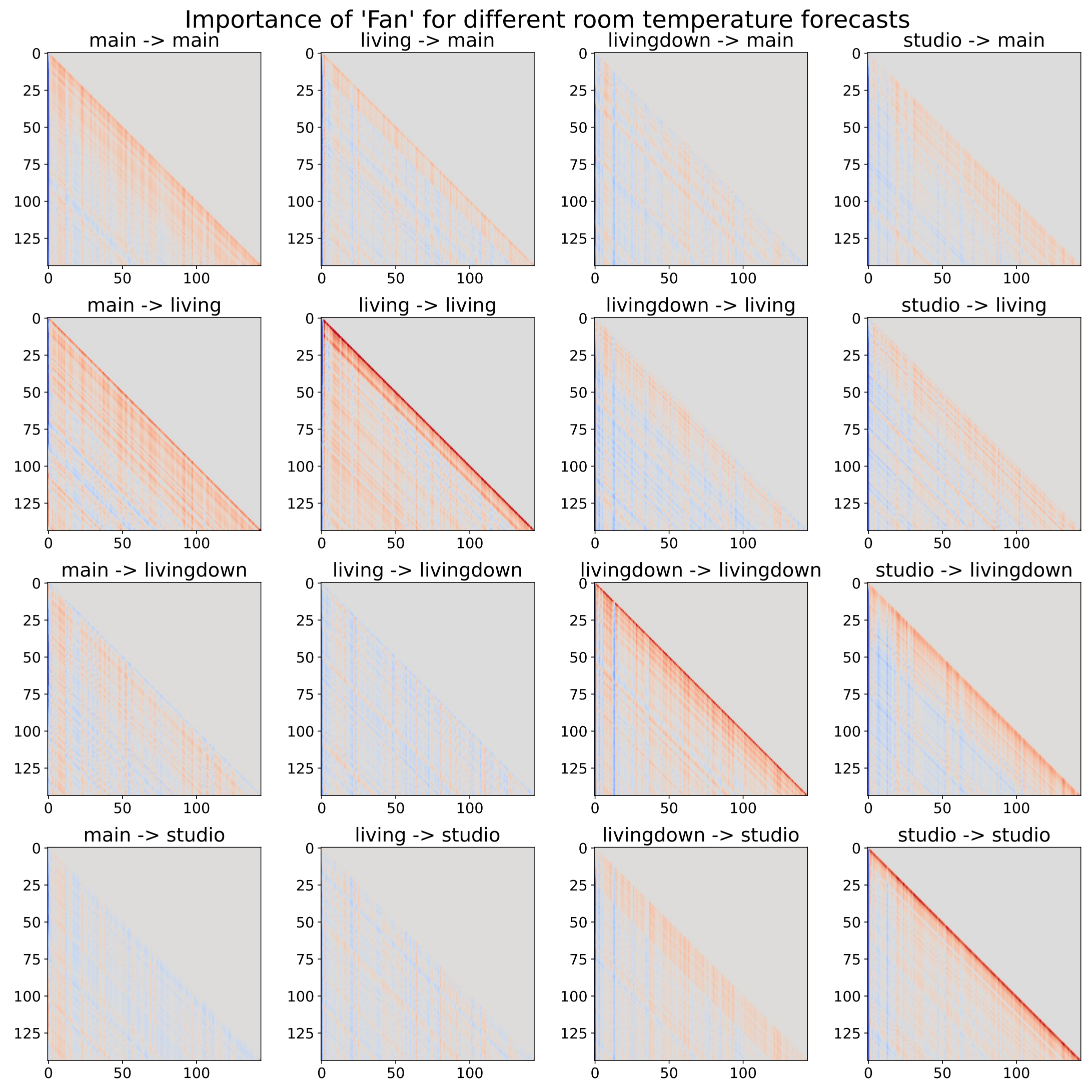

Figure 4.17: Model weight magnitudes for the ’Fan’-setting for the heat pump units in each room for all forecast horizons (along the y-axis) at all future time steps (along the x-axis). The colors red and blue signify positive and negative values respectively. Zero-valued weights are represented by the color gray. The model is the deterministic direct linear model trained on 50 weeks of data.

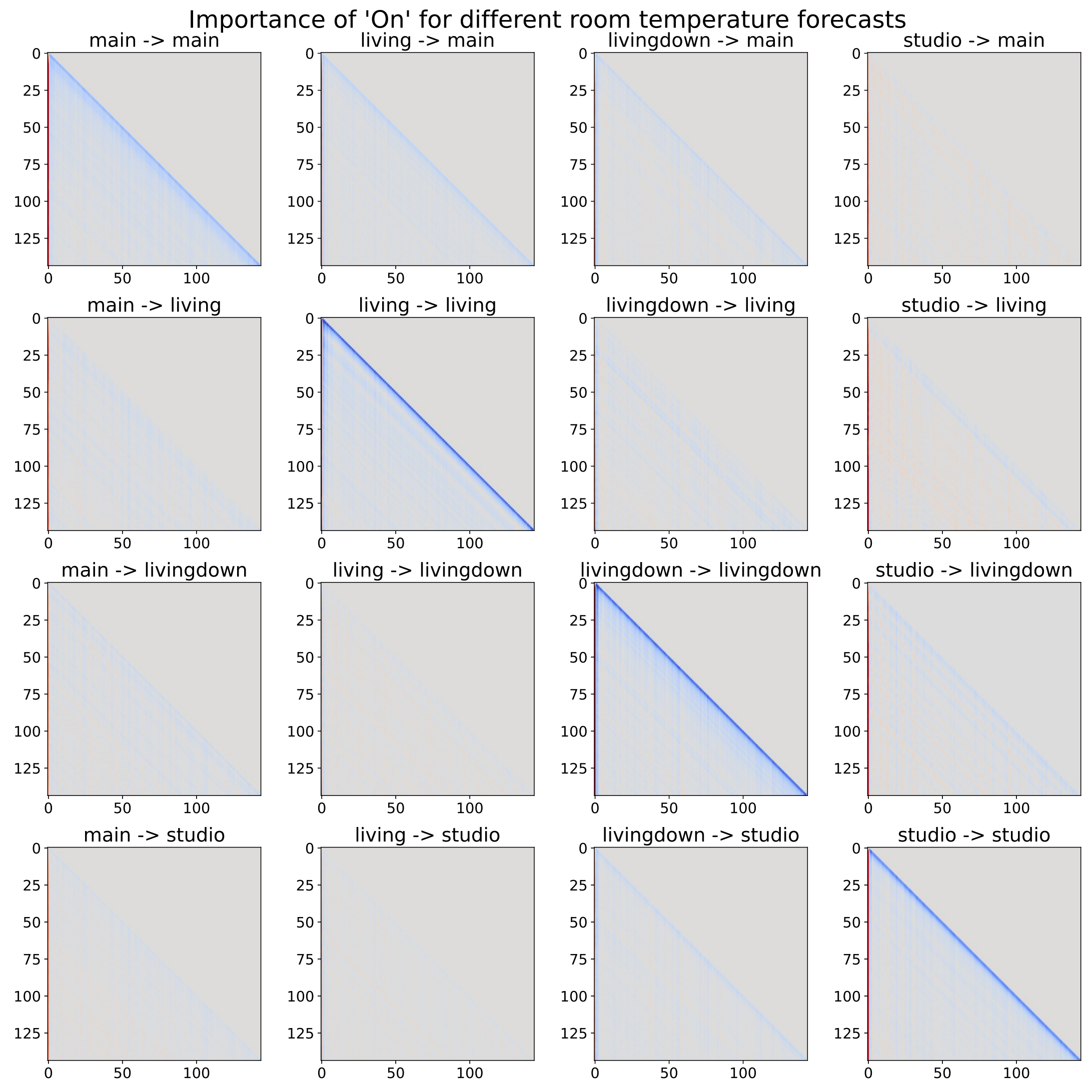

Figure 4.18: Model weight magnitudes for the ’On’-setting for the heat pump units in each room for all forecast horizons (along the y-axis) at all future time steps (along the x-axis). The colors red and blue signify positive and negative values respectively. Zero-valued weights are represented by the color gray. The model is the deterministic direct linear model trained on 50 weeks of data.

5

Power Consumption Models

In this chapter, two model configurations for predicting the power consumption of heat pumps are presented and evaluated. The predicting capabilities of a linear model are compared to that of a Deep Neural Network.

5.1 Modeling Power Consumption Dynamics

The price of power consumption is given at an hourly rate. To minimize the cost of heating while maintaining a desirable indoor temperature, it makes sense for a power consumption model to predict the hourly consumption. In addition, only the total power consumption for all units is available, which makes exploiting potential temporal power dynamics in single heat pumps difficult - especially on an hourly basis. Theoretically, a sudden change in a heat pump setting or the room temperature should affect the power consumption of a heat pump more than previously measured power usage. This is shown in Figure 3.2 and 3.5 which demonstrate little temporal dynamics. Static models that utilize heat pump settings and other relevant data for a time t are chosen to predict the power consumption for the same time.

A change in high power consumption has a greater impact on power consumption expenses than changes in very low power consumption. The stochastic power consumption predictor should preferably maintain a similar uncertainty for high power consumption predictions as for low ones. Data presented in section 3.1.1 shows that most of the hourly power consumption is very close to zero, which might make this challenging to achieve.

We can divide the relevant independent variables used to predict the hourly power consumption into three categories:

1. **Individual heat pump and room parameters** which only affect each respective heat pump. These consists of *Set*, *Fan*, and *Temp* for each of the rooms *main, living, living-down* and *studio*.

2. **Shared parameters** which may affect all heat pumps differently and should be treated as such. The relevant parameters here are the outside temperature and solar radiation (GHI). A constant 1 is also added to account for bias.

3. **Individual heat pump on/off-state** which can be used to determine whether the estimated power consumption of one heat pump should be added to the total consumption.

Table 5.1: Independent variables per room for power prediction.

Name	Category	Explanation
Set	Individual	Target temperature for the heat pump in a given room
Fan	Individual	Fan level for the heat pump in a given room
Temp	Individual	The actual temperature in a given room
AirTemp	Shared	Outside temperature from Solcast
GHI	Shared	Solar radiation from Solcast
1	Shared	Term to account for bias
On	on/off state	Whether the heat pump in a given room is on or not

If a heat pump is off, the heat pump should not consume power.

All the independent variables and their category is summarized in Table 5.1 for *main, living, livingdown* and *studio*. Note that if the power prediction model is used to predict the power consumption in the future, the room temperatures are unknown, and forecasts from the temperature forecasting models must be used. The same goes for air temperature and solar radiation.

These parameters are all measured at five minutes intervals, while the desired power consumption prediction period is an hour. The power consumption predictor should accumulate the predicted power consumption at each time step to produce a prediction for the total power consumption for that hour. Gaps in the measured data could corrupt the trained models, so only collections of time steps where all independent variables within an hour are present are used. This ensures a dataset with only valid data points.

5.2 Model Architectures

The proposed stochastic model for estimating the hourly power consumption consists of four similar modules, one for each indoor heat pump unit. In the following notations and equations, a deterministic model is achieved by omitting the "q"-notation. Each module consists of a function $f_q^u(\boldsymbol{x}_t^u, \boldsymbol{z}_t)$ which calculates the power consumption for the heat pump unit u at time step t and quantile q. $\boldsymbol{x}_t^u$ denotes the individual variables and $\boldsymbol{z}_t$ the shared variables as detailed in Table 5.1. The predicted total hourly power consumption $\hat{y}_q$ for all heat pumps at quantile q is the sum of power consumption for all heat pump indoor units $U = \{main, living, livingdown, studio\}$ and the time steps in an hour $T = [1, ..., 12]$:

$$\hat{y}_q = \sum_{u \in U} \sum_{t \in T} \max\left(s_t^u \cdot f_q^u(\boldsymbol{x}_t^u, \boldsymbol{z}_t), 0\right), \tag{5.1}$$

The model function f_q^u details the relationship between the independent variables and the total power consumption for unit u for five minutes after time t. s_t^u is the on/off state of the heat pump as explained in Table 5.1. The on/off information in s_t^u is used to filter out predicted power consumption contributions when the heat pump is off. To make sure the importance of parameters is not weighted differently for each time step t, the model parameters for a unit u remain static for all times $t \in T$. Since the heat pump only may draw power, and not contribute power to the grid, a saturation mask is applied to prevent negatively predicted power consumption. The model function f_q^u decides the complexity of the model. In this Thesis, it is given by a linear model and a deep neural network.

5.2.1 Linear Model

The linear model function is given by:

$$f_q^u(\boldsymbol{x}_t^u, \boldsymbol{z_t}) = \boldsymbol{W}_q^u \begin{bmatrix} \boldsymbol{x}_t^u \\ \boldsymbol{z}_t \end{bmatrix}, \tag{5.2}$$

where $\boldsymbol{W}_q^u$ denotes the model weights for unit *u* at quantile q. This results in the full model:

$$\hat{y}_q = \sum_{u \in U} \sum_{t \in T} \max \left(s_t^u \cdot \left(\boldsymbol{W}_q^u \begin{bmatrix} \boldsymbol{x}_t^u \\ \boldsymbol{z}_t \end{bmatrix} \right), 0 \right). \tag{5.3}$$

Assuming $\boldsymbol{W}_q^{linear}$ contains all unit weights $\boldsymbol{W}_q^u$ for $u \in U$, the minimization problem to be solved that yields the optimal model weights is given by:

$$\hat{\boldsymbol{W}}_q^{linear} = \underset{\boldsymbol{W}_q^{linear}}{\arg\min}\, L(\boldsymbol{W}_q^{linear}) \tag{5.4}$$

where the loss function $L(\boldsymbol{W}_q^{linear})$ is given by the MSE of all samples for the deterministic model and MAD for the stochastic one.

The sum of power predictions for all time steps in addition to the saturation function and the on/off multiplication makes least squares optimization an unfeasible method for solving the minimization problem. Solving (5.4) is performed using gradient descent with randomized initial weights. Both deterministic and stochastic models are trained by employing the respective loss functions. Separate models are created for each of the quantiles $q \in \{0.1, 0.5, 0.9\}$ for the stochastic case.

The learning rate is the main tunable parameter. This model is very sensitive to the learning rate - especially the stochastic model. The window of learning rates that could be used to solve the minimization problem was quite small. With a too high learning rate, the solution diverges, while with a too low learning rate, the solution does not converge fast enough - even with an increase in the number of training epochs. The learning rates for the deterministic and stochastic models are tuned to 10^{-5} and 10^{-8} respectively. When the learning rate is set, training times vary with the size of the training set. When using one week of data, a single model is trained in approximately ten seconds on a decent CPU. With fifty weeks of data, the time needed for convergence increases to one minute.

5.2.2 Deep Neural Network Model

By replacing the linear relationship presented in the previous section with the more complex feed-forward deep neural network, the model should be able to learn more complex dynamics in the data. A deterministic feed-forward DNN model function is given by:

$$f^u(\boldsymbol{x}_t^u, \boldsymbol{z}_t) = \text{DNN}^u(\boldsymbol{x}_t^u, \boldsymbol{z}_t). \tag{5.5}$$

where DNN^u denotes a deep neural network with the individual unit and shared parameters as inputs. The output of the network is a prediction for the power consumed by unit u at time t. The full deterministic deep neural network model is provided by:

$$\hat{y} = \sum_{u \in U} \sum_{t \in T} \max \left(s_t^u \cdot \text{DNN}^u(\boldsymbol{x}_t^u, \boldsymbol{z}_t), 0 \right). \tag{5.6}$$

Assuming $\boldsymbol{W}$ contains all model weights in all DNN^u for $u \in U$, the optimal set of weights is given by:

$$\hat{\boldsymbol{W}} = \arg\min_{\boldsymbol{W}} L(\boldsymbol{W}) \tag{5.7}$$

The cost function $L(\boldsymbol{W})$ consists of a loss function and an L2 regularization term to increase generalizability as described in section 2.4. The loss function is the MSE for the deterministic model, and the total cost to be minimized is:

$$L(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{w \in \boldsymbol{W}} ||w||^2, \tag{5.8}$$

where w is a single weight in $\boldsymbol{W}$ and λ weights the regularization term.

A stochastic model is created by changing the number of DNN outputs from one to the desired number of quantile predictions for each unit u. The chosen quantiles are $Q = \{0.1, 0.5, 0.9\}$. With $\boldsymbol{W}$ containing all stochastic model weights, the loss function for the stochastic model is the sum of MAD over all quantiles. The cost function $L(\boldsymbol{W})$ for the stochastic model also consists of a loss function and an L2 regularization term:

$$L(\boldsymbol{W}) = \sum_{q \in Q} MAD(\boldsymbol{y}, \hat{\boldsymbol{y}}_q, q) + \lambda \sum_{w \in \boldsymbol{W}} ||w||^2, \tag{5.9}$$

where $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}_q$ consists of all true and predicted target values at quantile q. A prediction at quantile q is given by:

$$\hat{y}_q = \sum_{u \in U}\sum_{t \in T} \max\left(s_t^u \cdot \mathrm{DNN}^u(\boldsymbol{x}_t^u, \boldsymbol{z}_t)[q], 0\right). \tag{5.10}$$

The activation function chosen in these networks is the ReLU as it combats the vanishing gradient problem. The mini-batch stochastic gradient descent method Adam is chosen as the optimization method since it is proven to be more efficient than other gradient descent methods for deep learning problems. The models are trained for a certain amount of epochs. the number of epochs is tuned as a hyperparameter.

All independent variables are scaled using min-max normalization to be in the range of $[0, 1]$. Most of the variables do not have a gaussian distribution, and this method is thus preferred over the standardization technique. As the ReLU activation function is used for each node in the network, the dependent variable is also scaled using min-max normalization.

The tunable hyperparameters in these models are the number of layers, number of neurons per layer, learning rate, and the regularization weight in addition to the number of training epochs. The model is highly sensitive to different hyperparameter configurations, with some sets of parameters yielding unusable models. The number of hidden layers and neurons are tuned by iteratively increasing them from low initial values until the performance starts to decrease. At the same time, the learning rate and regularization weight are tuned together with the number of epochs to achieve convergence of the cost function. As the deterministic and stochastic DNN models have a different number of outputs, the resulting architecture parameters differ. Table 5.2 contains the parameter values that yielded the best models found during the tuning procedure.

Table 5.2: Hyperparameters for DNN power model.

Parameter	Deterministic models	Stochastic models
Hidden layers	2	3
Neurons per hidden layer	5	10
Learning rate	0.001	0.001
Regularization weight	0.001	0.001
Epochs	100	100

With the given parameters, training the models on one week of data takes approximately one minute while using fifty weeks of data increases the training time to forty minutes. This is a large increase from the linear models, but it is still feasible to tune the hyperparameters in a short period and compare model performances without spending weeks training models.

5.3 Model Evaluations

This section presents an evaluation of the deterministic and stochastic linear and DNN-based models on the test data. The linear models were trained on 20 training data sets ranging from one to fifty weeks, while the DNN ones were trained on one and fifty weeks of data. The stochastic models are trained to output predictions for quantiles $q \in \{0.1, 0.5, 0.9\}$.

5.3.1 Linear Models

The mean absolute errors and expected errors on test data for all trained linear models are presented in Figure 5.1. The performance of the stochastic and deterministic models are quite similar with the difference in error too small to matter. Both models trained on one week of data outperforms all other with an MAE of under 0.2 kWh. The model needs to be trained on thirty weeks of data to reach similar performance. The expected errors represent the mean errors for each model on the test data. All trained models have a tendency to predict slightly higher power consumption than the true ones except for the model trained for thirty weeks. For most of the trained models, the overshoot is under 0.1 kWh. While the model trained on fifty weeks of data has a lower MAE than the one trained on six weeks of data, the expected error of the six-week model is closer to zero than the other one. The expected errors for the upper and lower quantile outputs of the stochastic model show that the one-week model is more certain than the other ones. The models trained on the most data are among the most uncertain models trained, but the variations for the different models are quite small.

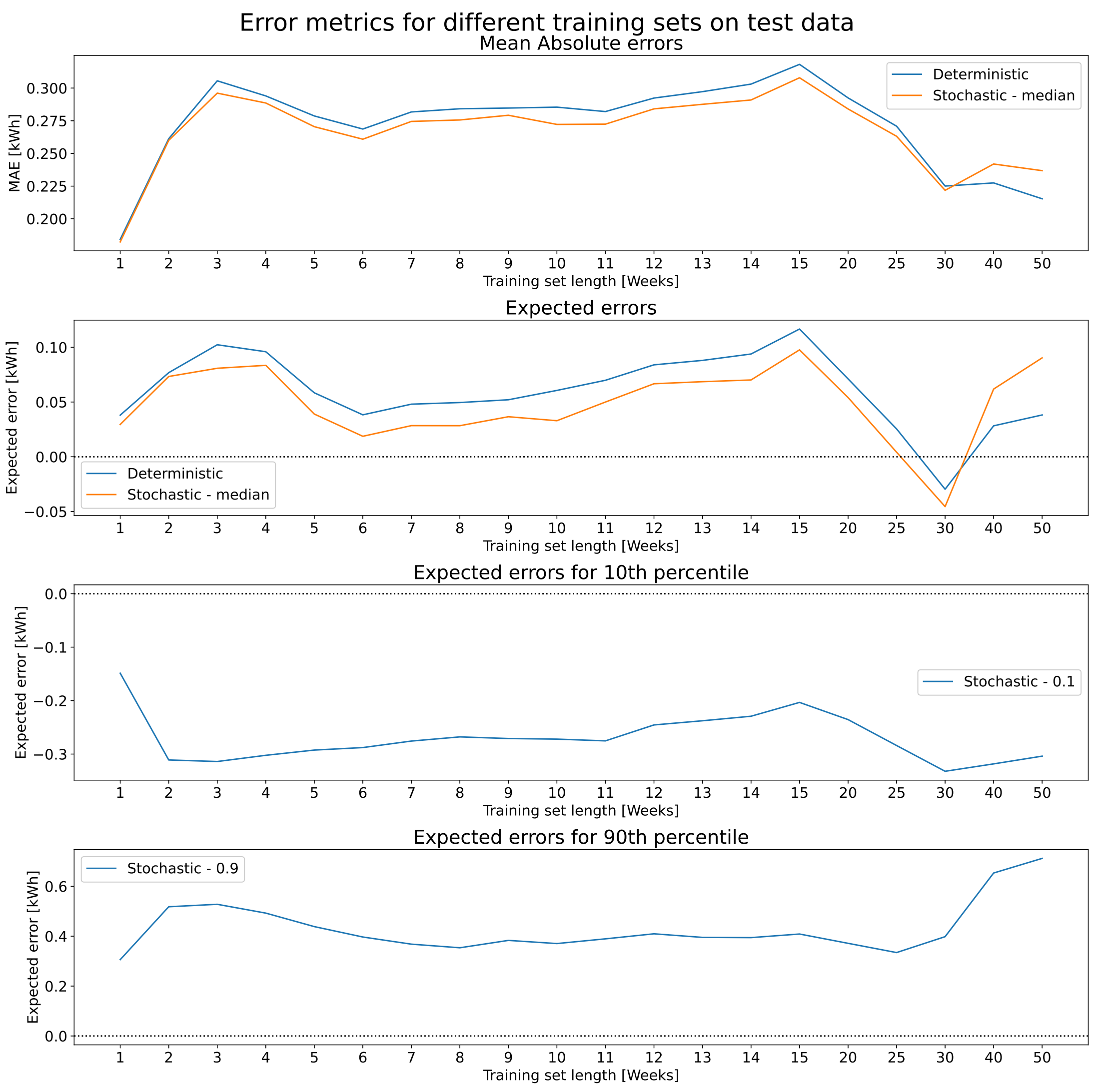

Figure 5.1: MAE and expected errors for deterministic and stochastic linear power prediction models trained on different training data.

Figures 5.2 and 5.3 present the residuals for the deterministic and median of the stochastic models respectively. These plots show that the models trained on one and fifty weeks of data performed better than the others on the test data. For most of the models, the predictions are biased with a tendency to predict too-high power consumption with the largest residuals being negative. This may explain the low expected errors for the models with the higher mean absolute errors. For the models trained on the most data, the residuals decrease in magnitude.

Figures 5.4 and 5.5 show the deterministic and stochastic predictions against the true power consumption. The median of the stochastic predictions behaves quite similarly to the deterministic ones. In the stochastic plots, the models are more certain for lower predictions than the higher ones. However, for most of the models, the sample errors do not diverge significantly. Note that there are few samples with power consumption higher than 3 kWh, so no conclusions can be drawn on the models' performance in this scenario. For most of the models, there are quite a few zero predictions while the true power consumption is up to 1 kWh. Only the one and fifty weeks models have few such predictions. It is these predictions that result in large model MAEs. Figure 3.2 shows a large part of the measured power consumption is close to zero. It might be that the model overfits this data which results in false zero-predictions. Note that with enough data, there is a reduction in this effect. There might also be some dynamics that the models do not account for sufficiently.

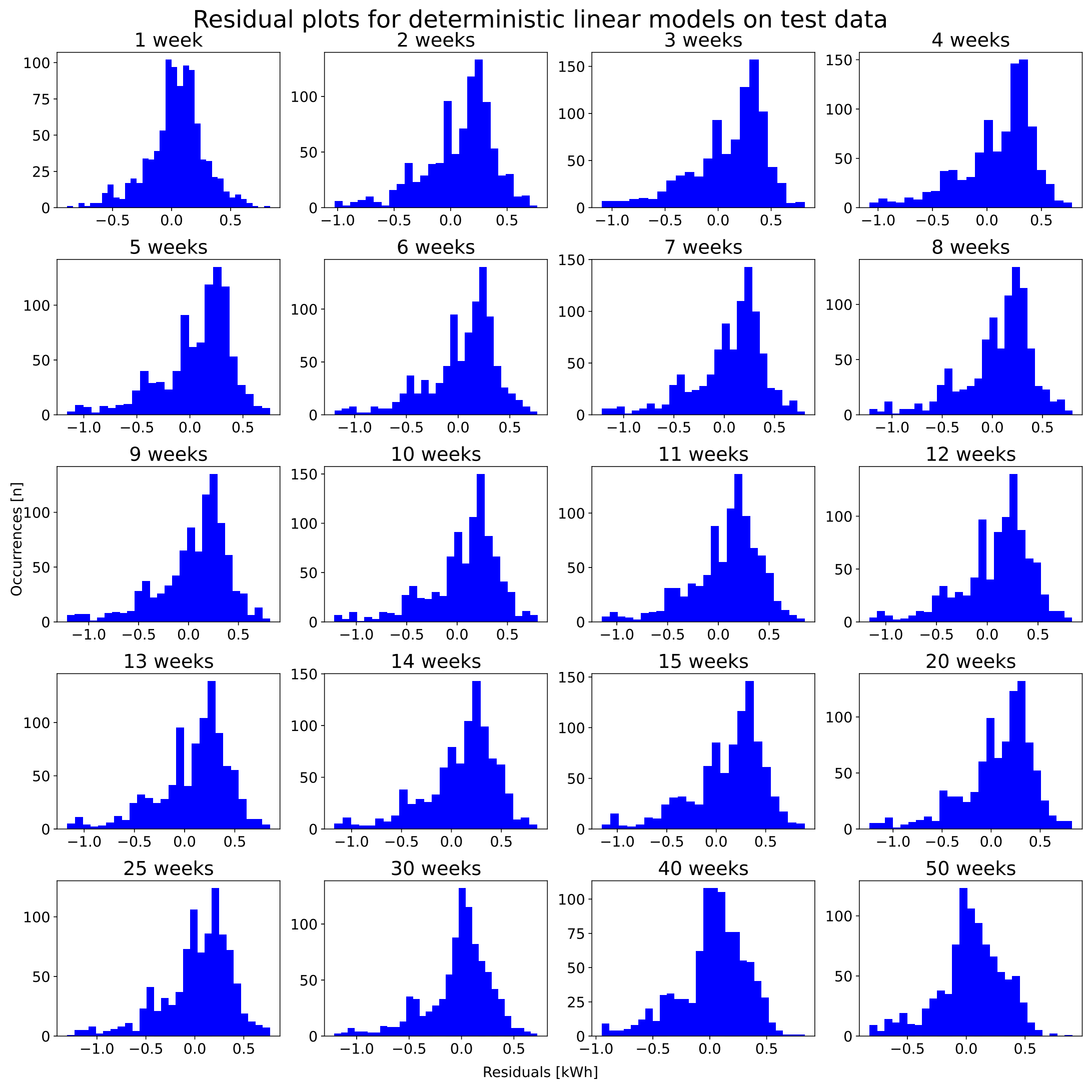

Figure 5.2: Residuals for linear deterministic power consumption models.

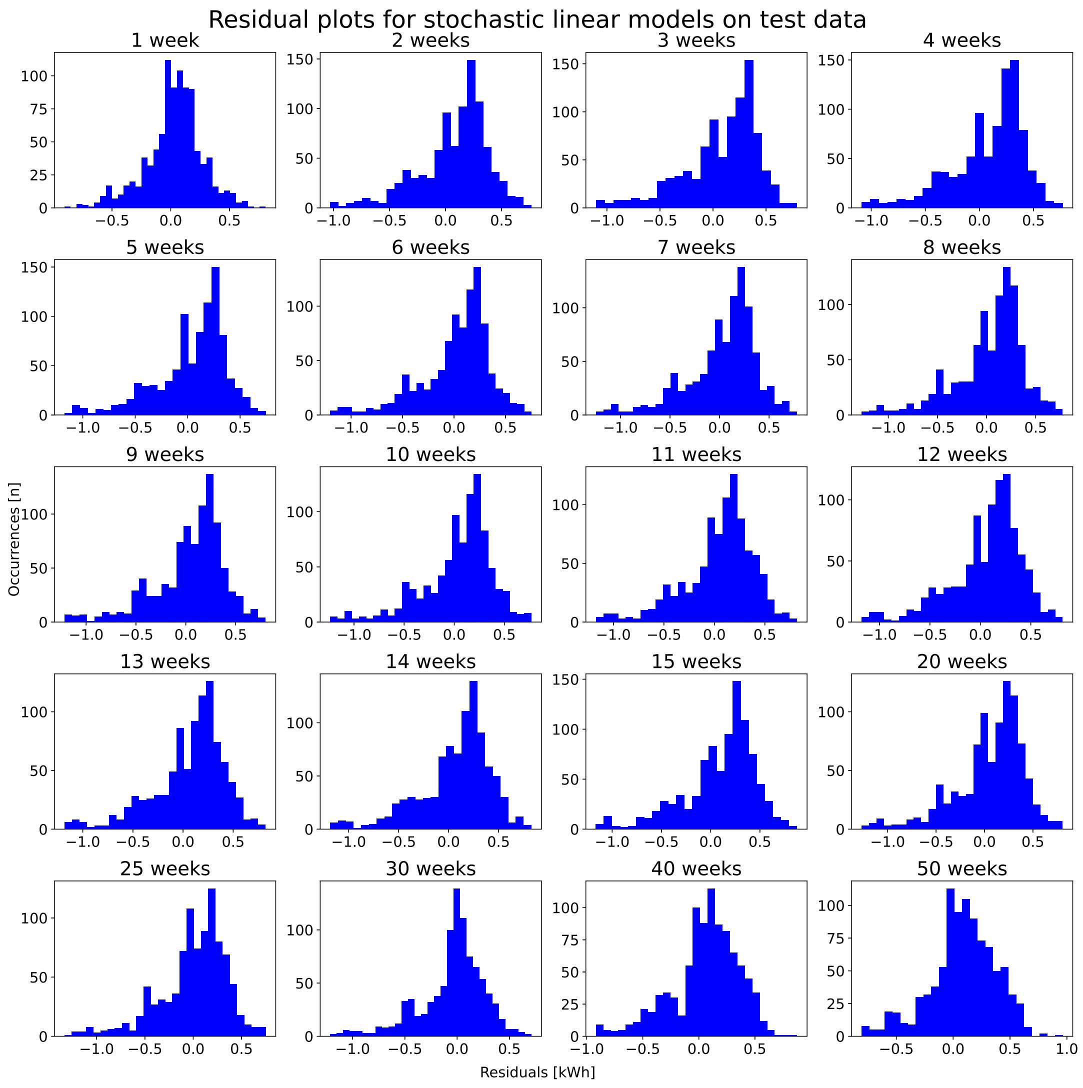

Figure 5.3: Residuals for the median of linear stochastic power consumption models.

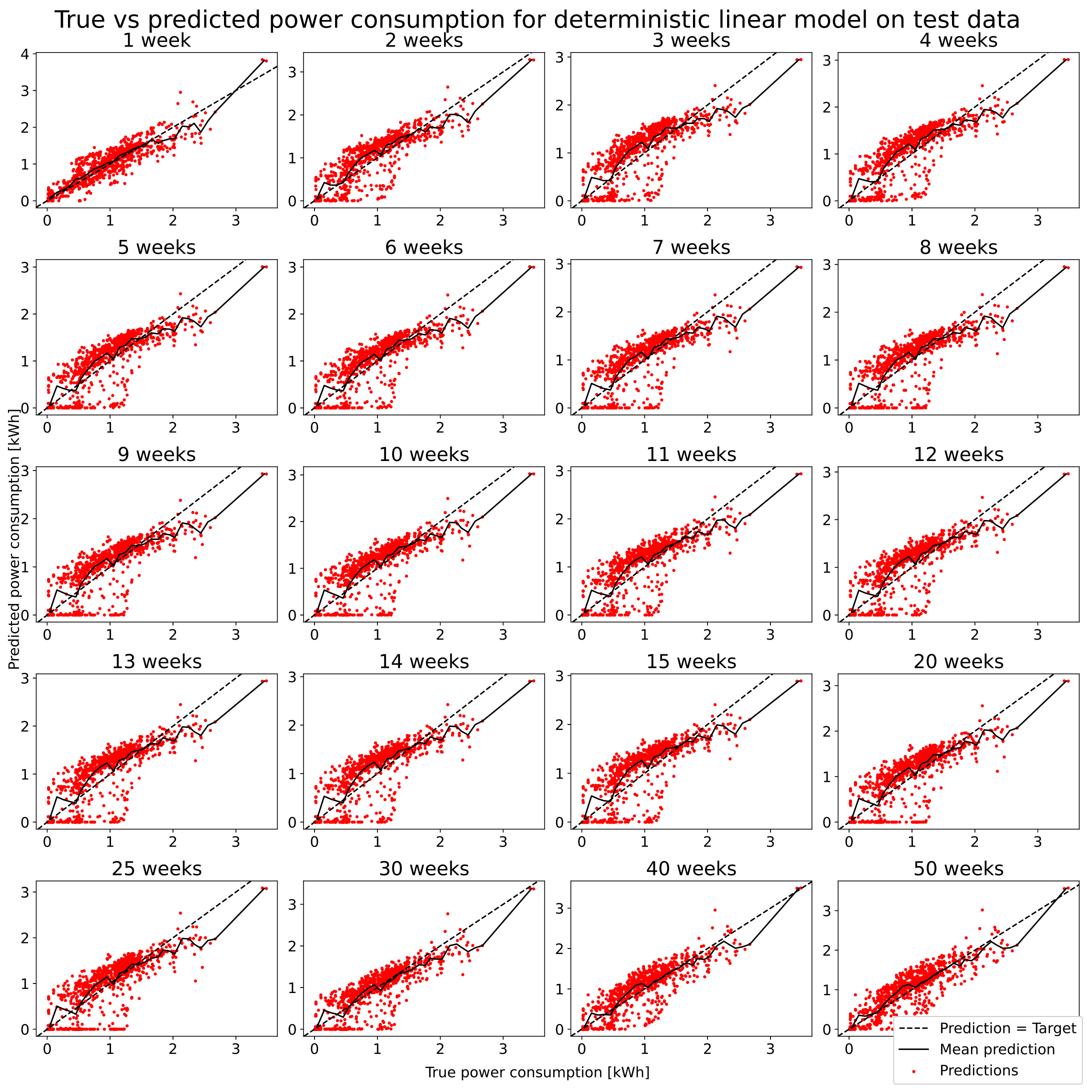

Figure 5.4: Deterministic predictions against the true power consumption. The red markers represent individual predictions, and the black line is the mean predictions with the dotted line representing true power consumption.

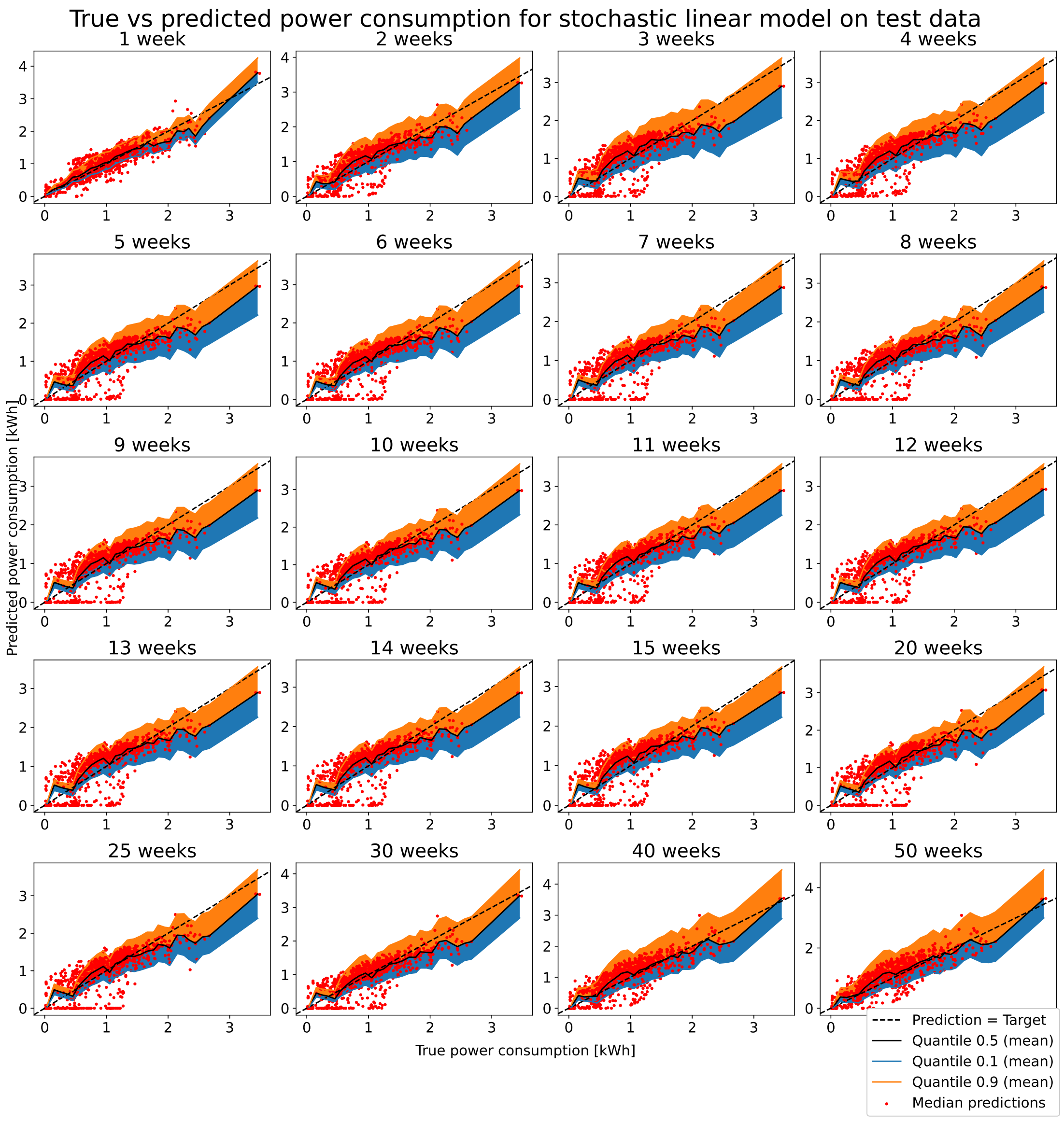

Figure 5.5: Stochastic predictions against the true power consumption. The red markers represent individual predictions, and the black line is the mean predictions with the orange and blue representing the quantile predictions for quantiles 0.9 and 0.1. The dotted line represents true power consumption.

5.3.2 Deep Learning Models

Using deep neural networks, deterministic and stochastic models were trained on one week and fifty weeks of data. The mean absolute errors and expected errors of the models are summarized in Table 5.3. The deep learning-based models trained on fifty weeks of data outperforms the ones trained on one week with the stochastic model having the lowest expected error.

Figure 5.6 presents the residuals for the deep learning-based models. These results confirm the improved performance of the deep learning models with more data. For the models trained on one week of data, the residuals are higher compared with the linear models. With fifty weeks of data, the residuals for the deep learning-based models are similar to the linear models. Figure 5.7 shows the deterministic and stochastic predictions of the deep learning models against the

Table 5.3: Error measures for DNN-based power prediction models.

Deep Learning based model	MAE	Expected error
Deterministic - 1 week	0.38 kWh	0.085 kWh
Stochastic (median) - 1 week	0.35 kWh	-0.201 kWh
Deterministic - 50 weeks	0.23 kWh	-0.015 kWh
Stochastic (median) - 50 weeks	0.23 kWh	0.001 kWh

true power consumption. The predictions of the stochastic deep learning model trained on one week have quite a large spread with many 0.5 kWh and 0 kWh predictions. The higher the true power consumption, the larger the spread. The deterministic models also have a large spread in the predictions, but the deviations are not quite as significant. The stochastic model trained on fifty weeks of data shows some consistency for low power consumption, but struggles at higher power consumption levels.

The stochasticity of DNN training procedures detailed in section 2.4 greatly influences the training of these deep neural network-based models. Training the presented deep learning models several times results in widely different model performances. The MAE of the stochastic model trained on fifty weeks of data ranges from 0.14 kWh to 3 kWh with some models performing very well and some having large biases. This is a symptom of the training procedure struggling with fitting the model parameters to the data.

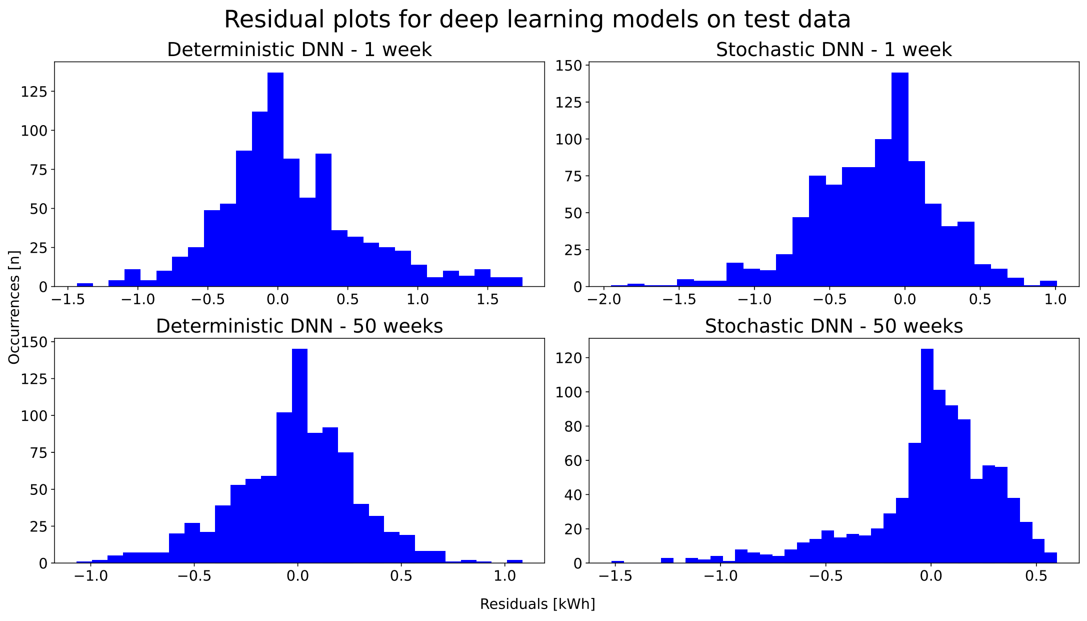

Figure 5.6: Residuals for deep learning-based power consumption models.

Table 5.4: Ranges and fixed setting values for input-output parameter analysis.

Parameter	Fixed value	Range	Unit
Set	20	15 - 32	$°C$
Fan	1	1 - 5	Level
Temp	20	9 - 32	$°C$
AirTemp	0	-15 - 30	$°C$
GHI	0.2	0 - 1	W/m^2

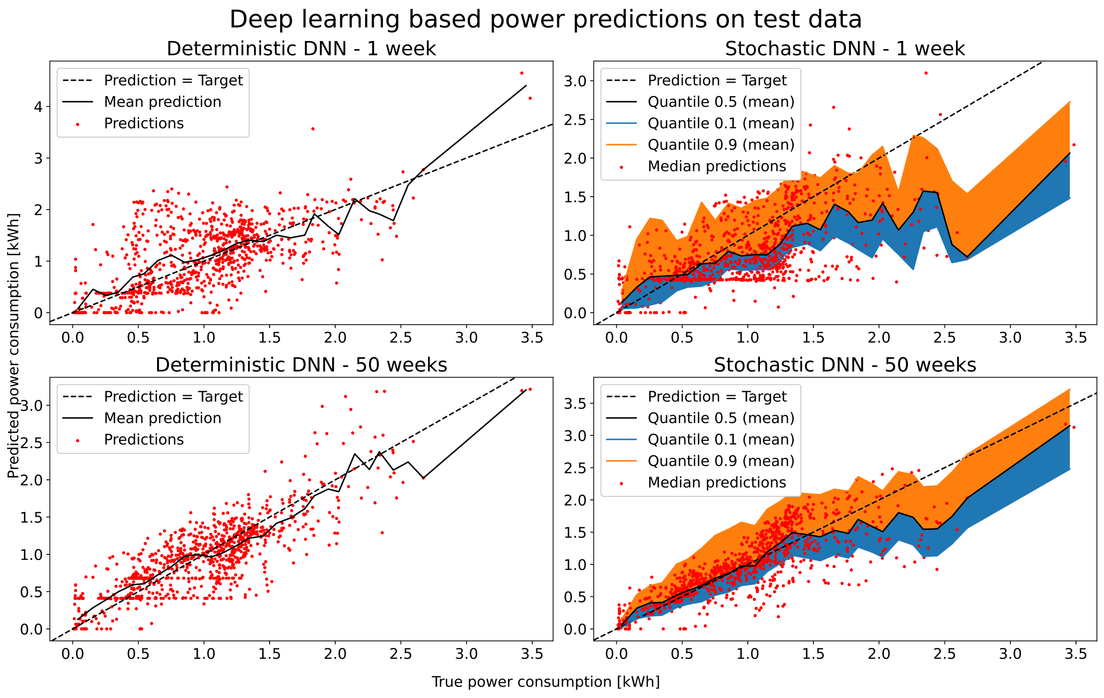

Figure 5.7: Deterministic and stochastic predictions against the true power consumption for deep learning models. The red markers represent individual predictions, and the black line is the mean predictions with the orange and blue representing the quantile predictions for quantiles 0.9 and 0.1. The dotted line represents true power consumption.

5.4 Model Analysis

The input-output correlations of each input parameter in the deterministic linear and deep learning models were analyzed. This was done by setting all parameters to a fixed common value and changing each parameter for each heat pump in a given range. The fixed values and their ranges are presented in Table 5.4. When considering one parameter, contributions from other heat pumps are masked out by setting the relevant on/off settings to zero.

Figure 5.8 presents the results for the linear deterministic model trained on fifty weeks of data. As expected, the predicted power consumption increases with increasing ”Set”-setting for all heat pumps and decreases with decreasing room temperatures. The increase and decrease are quite similar for the different heat pump units, but the magnitude is quite different. As the true power consumption of each heat pump is unknown, it is difficult to assess whether the difference in these dynamics is true. The predicted power consumption also decreases with increasing air temperature, until it is saturated at zero. This happens at different temperatures for the different rooms. An outside temperature of above $15°C$ has no effect on the predicted power consumption for *livingdown*. This may be a part of the cause for the zero predictions. Figures 5.9 and 5.10 present the same correlations for deterministic linear models trained on one and six weeks of data respectively. There is no significant saturation for the one-week model, while both *Temp* and *AirTemp* are saturated at common levels for some of the rooms for the six weeks model. The parameter slopes of the six weeks model are significantly more gentle than the two other correlations presented. The model trained on six weeks of data have several more zero predictions than the two others, and these phenomena are probably some of the reasons for this.

In all learned models, the Fan-level and GHI are given little influence. They remain constant at differing magnitudes for all units. This indicates that they are not very relevant for power consumption and that the training procedure struggles to find any correlation.

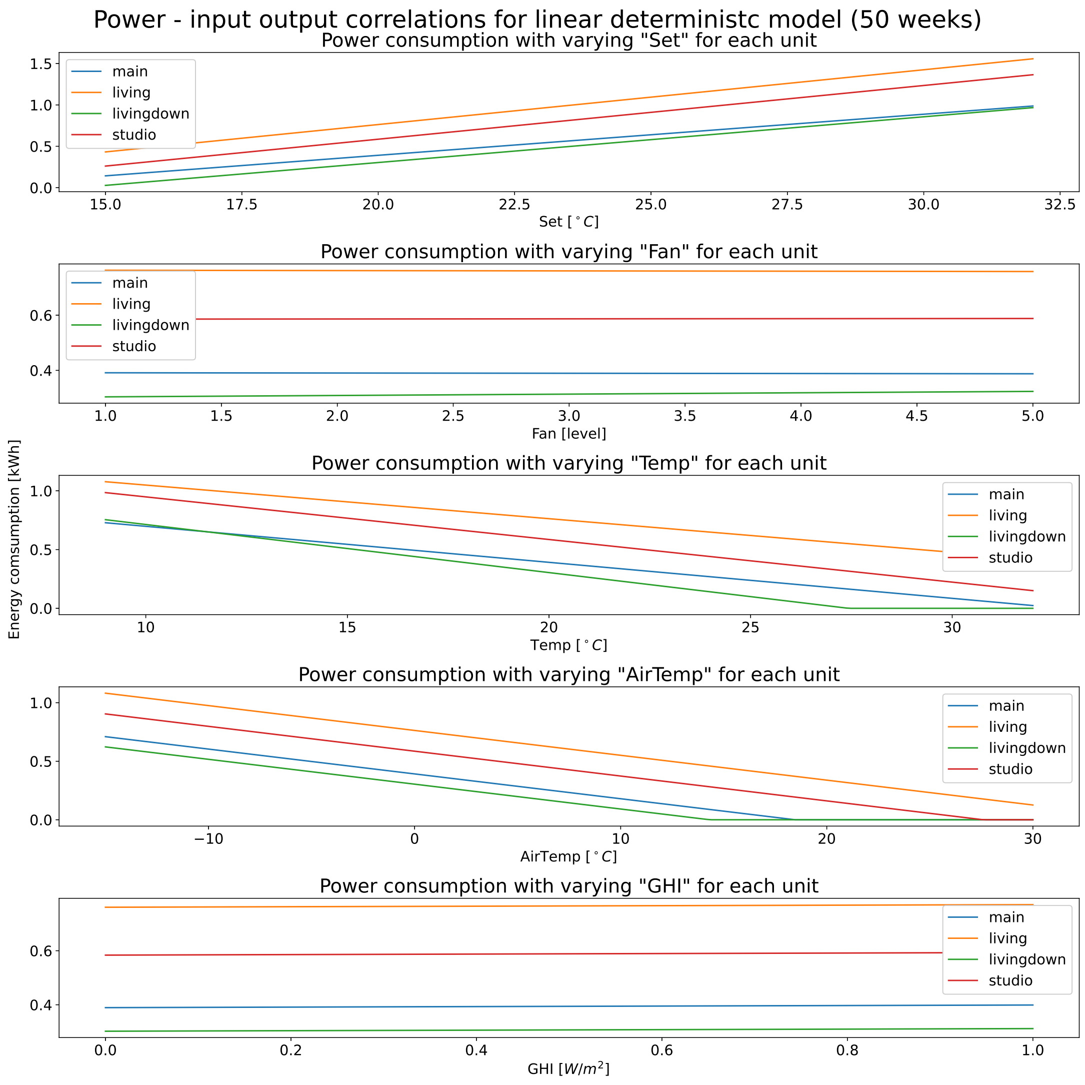

Figure 5.8: Correlations between input parameters and their contributions to the predicted power consumption for the linear deterministic model.

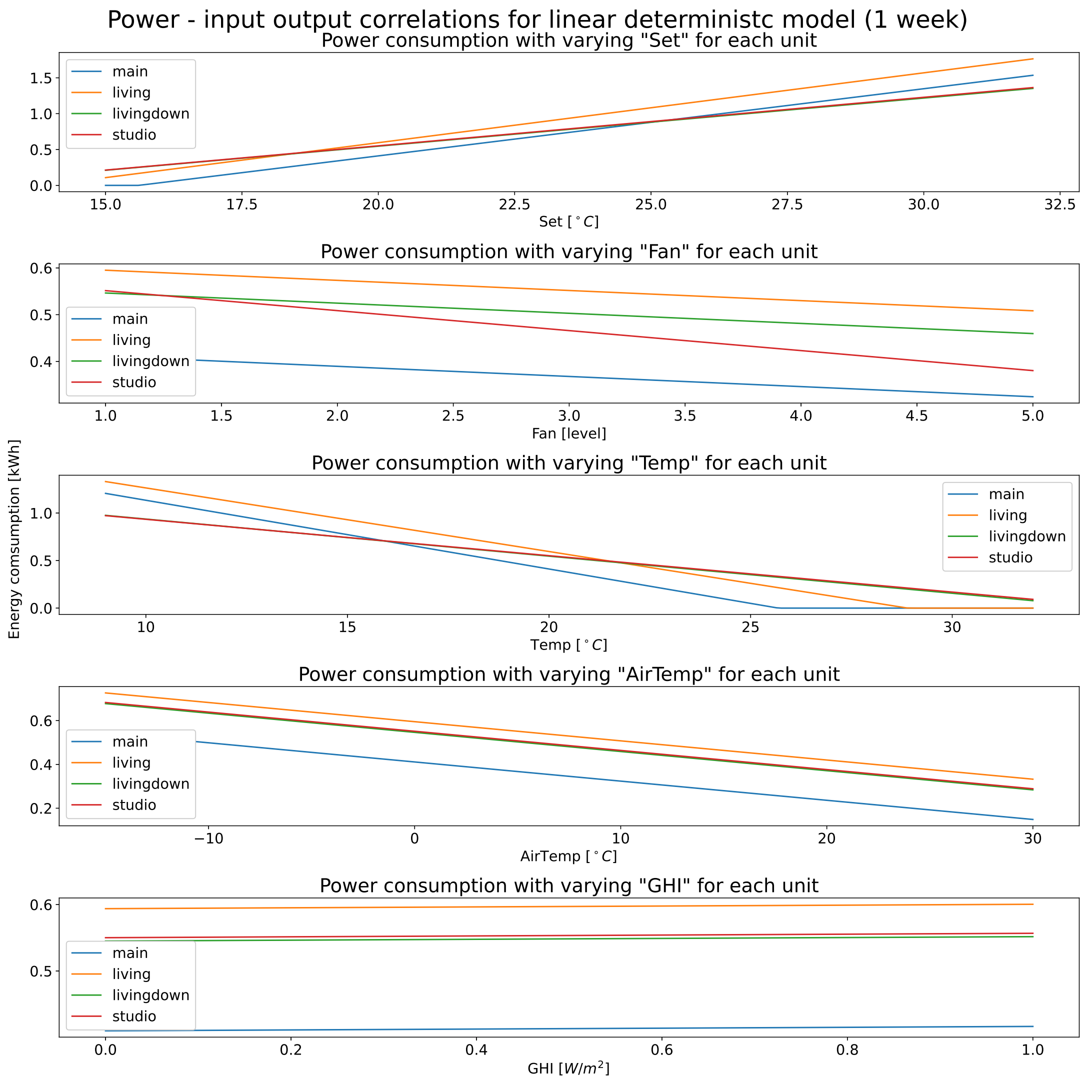

Figure 5.9: Correlations between input parameters and their contributions to the predicted power consumption for the linear deterministic model.

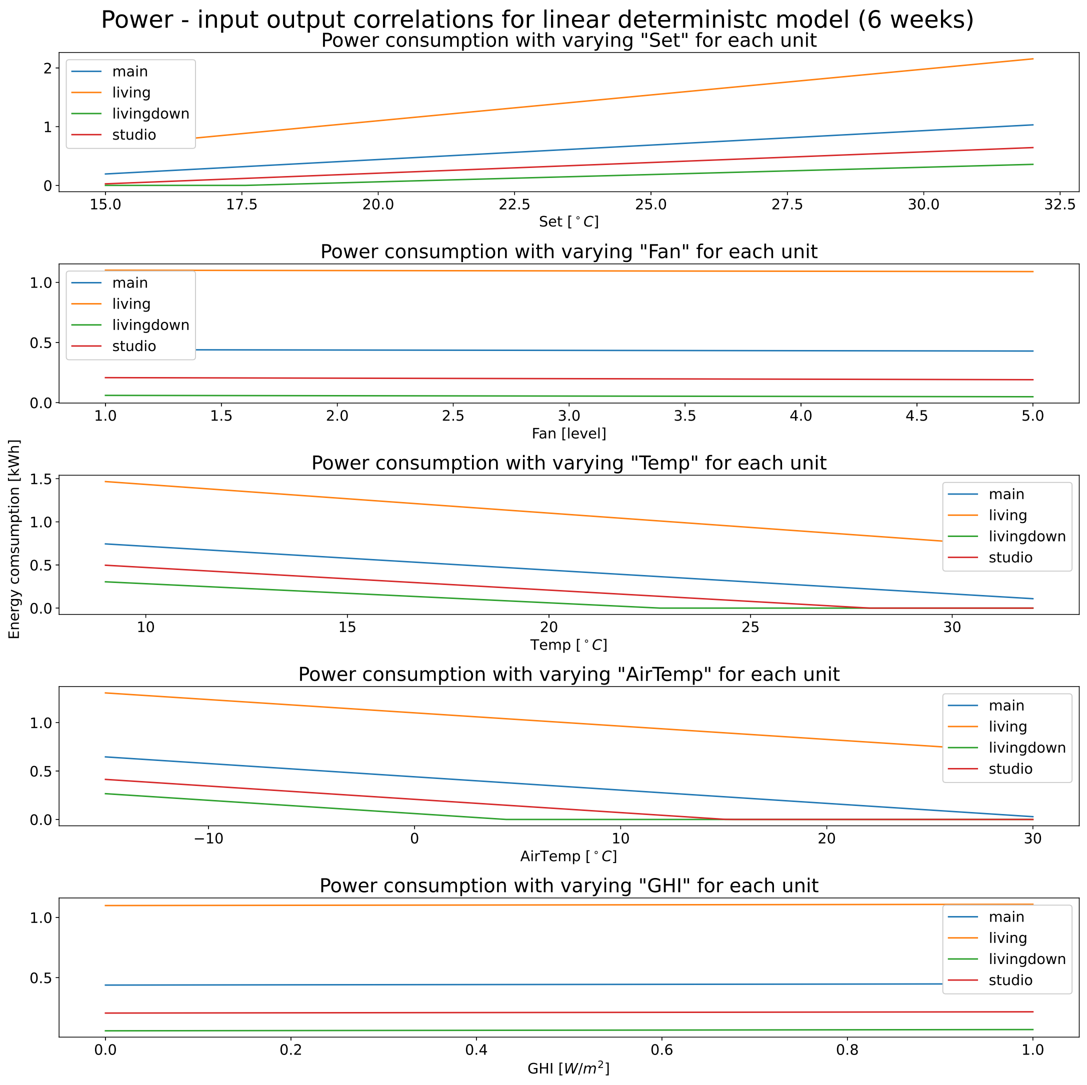

Figure 5.10: Correlations between input parameters and their contributions to the predicted power consumption for the linear deterministic model.

Figure 5.11 presents the same parameter analysis for the stochastic deep learning-based model trained on fifty weeks of data. There are some similar characteristics for the *Set*, *Temp*, and *AirTemp* parameters compared to the linear models. Note that the contributions from *main* and *studio* are non-existent. The neural network seems to ignore the two rooms, and only learns weights for the two others. This can explain the prediction behavior presented in Figure 5.7 which contains multiple predictions at a constant level. One reason for this may be that the designed network architecture does not manage to learn the weights properly.

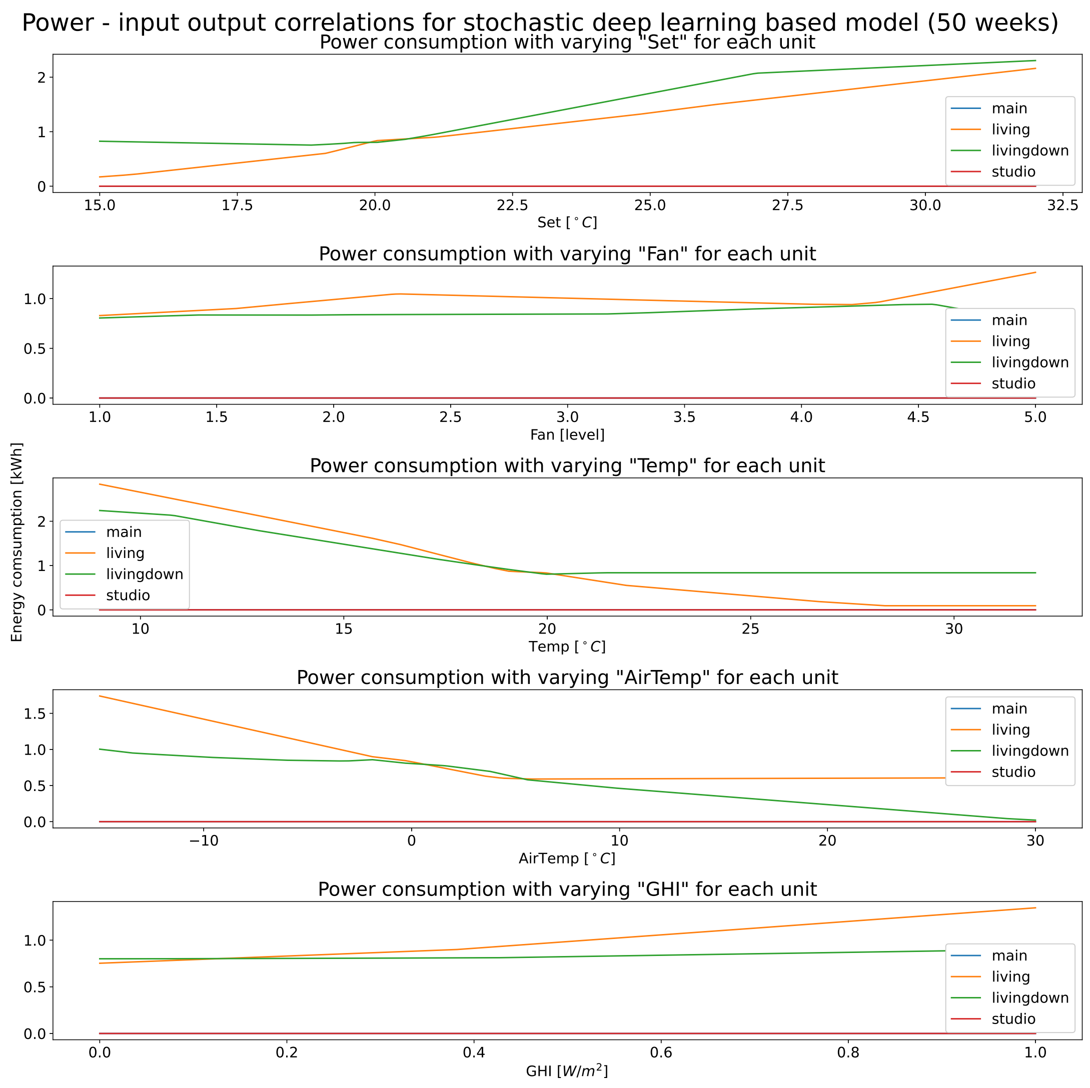

Figure 5.11: Correlations between input parameters and their contributions to the predicted power consumption for the linear deterministic model.

The static power prediction models assume that there are no correlations between predictions at different times. The prediction errors at different time steps should be independent of each other. This is not entirely the case. Figure 5.12 shows that there are some correlations between the errors for predictions at time t and $t-1$ for the deterministic linear model trained on fifty weeks of data. Figure 5.13 presents the autocorrelations for the error time series for 40 lagged time steps. The deep learning-based models also struggle to achieve low correlations between the prediction errors. Figures 5.14 and 5.15 present the joint distribution and autocorrelations for the errors produced by the stochastic deep learning-based model trained on fifty weeks of data. This autocorrelation could be a result of noise in the measurements that fluctuate slower than the measurements or that the data lacks some influential variable. Some misspecification in the model architecture could also result in autocorrelated errors. This observation reveals imperfections in the model that should be accounted for in further model development.

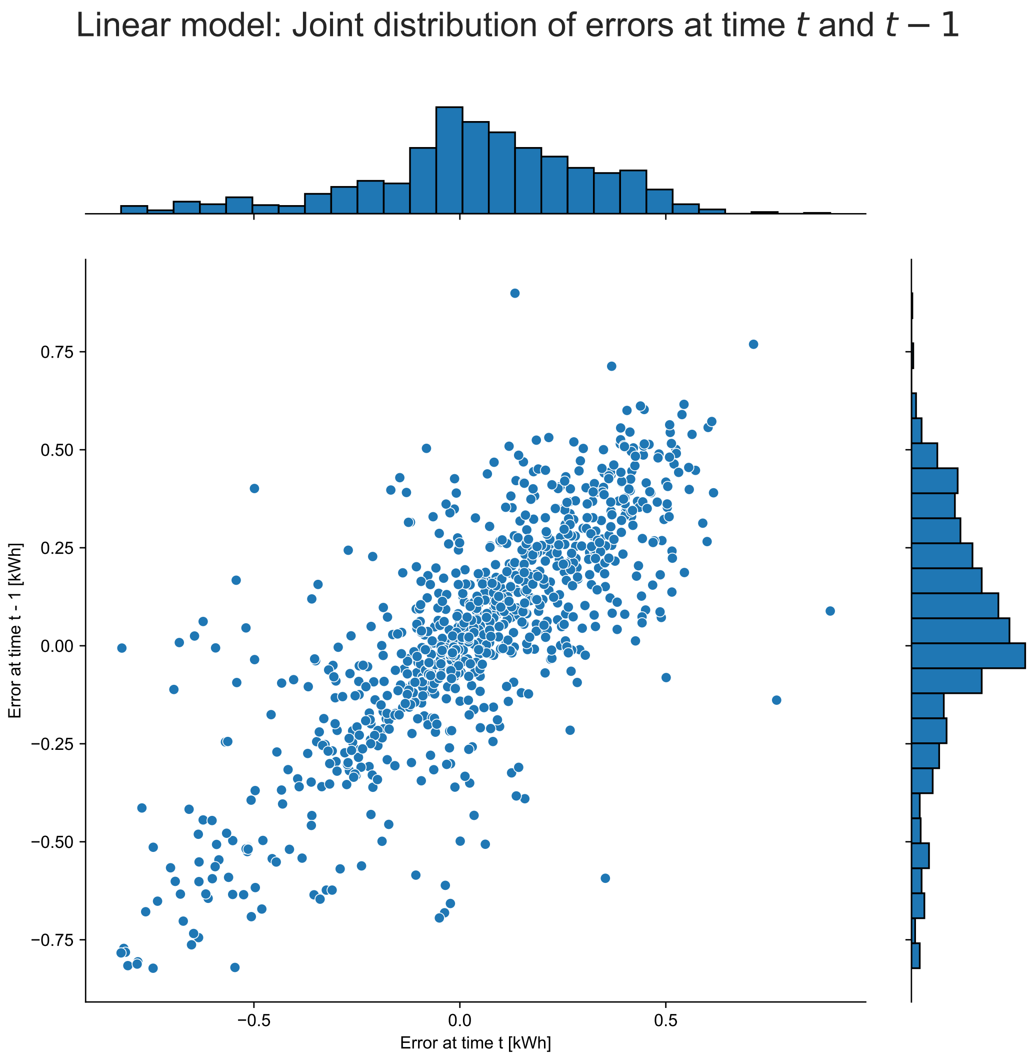

Figure 5.12: Joint distributions of errors at time t and $t-1$ from the deterministic linear model trained on fifty weeks of data.

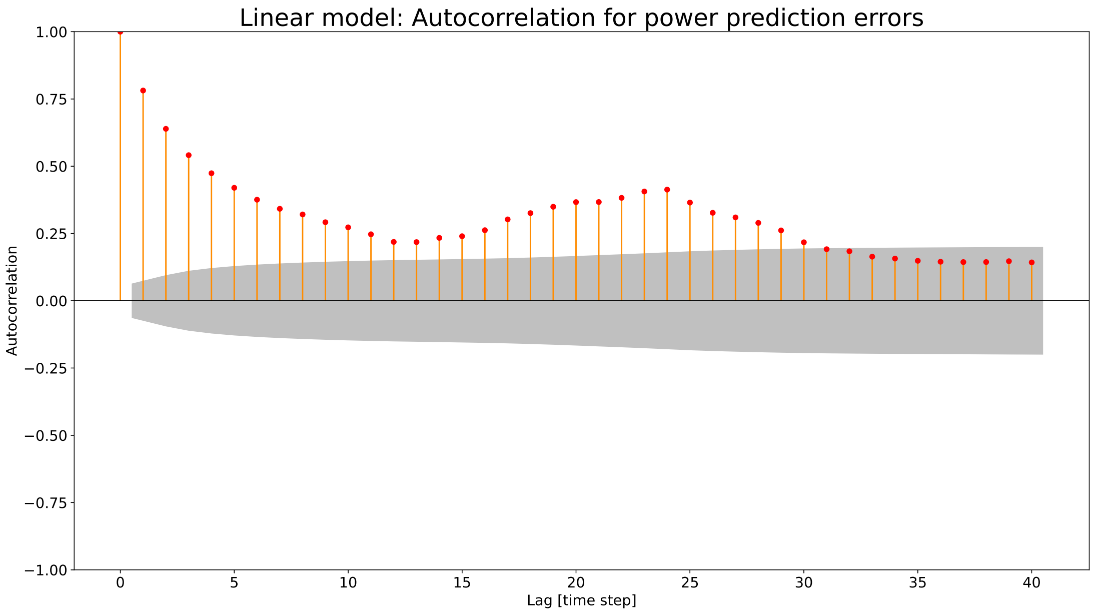

Figure 5.13: Autocorrelation for error time series from the deterministic linear model trained on fifty weeks of data.

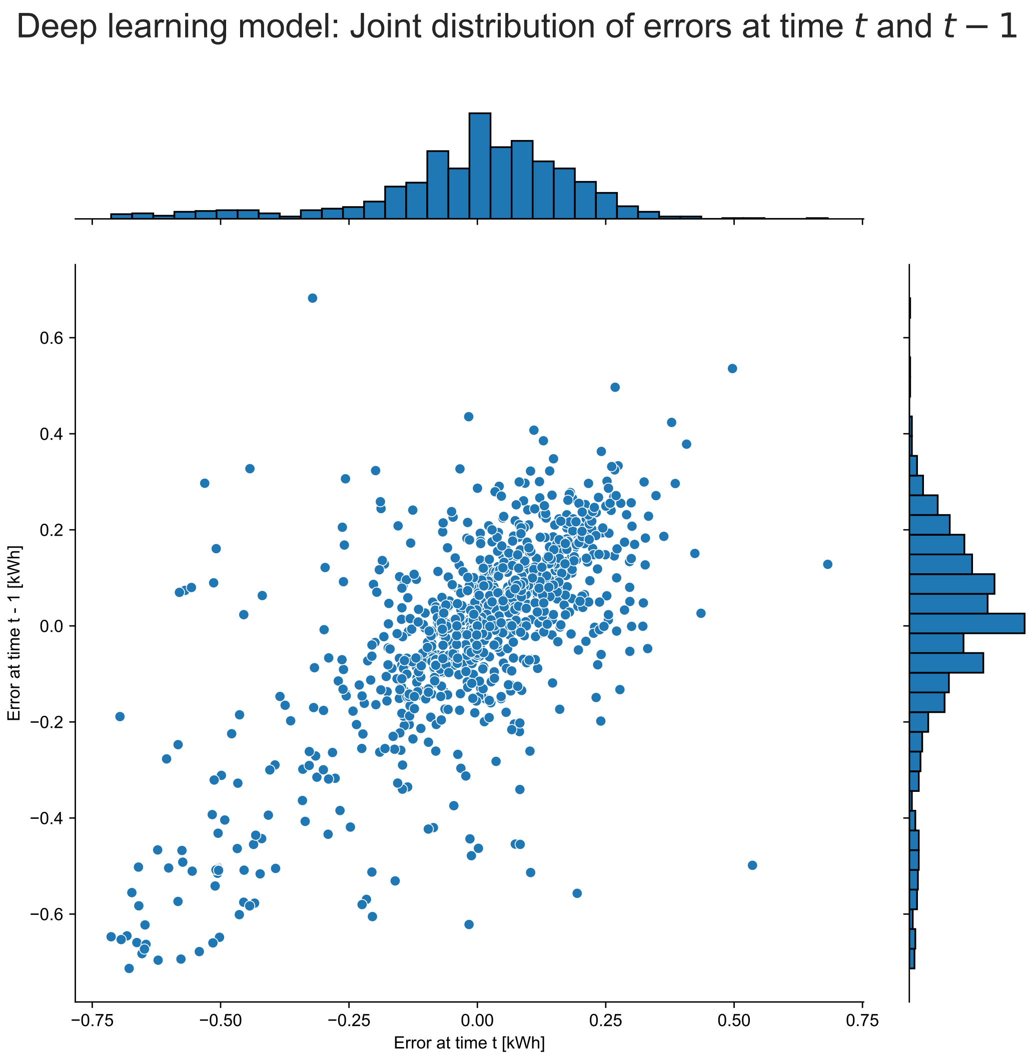

Figure 5.14: Joint distributions of errors at time t and $t-1$ from stochastic deep learning model trained on fifty weeks of data.

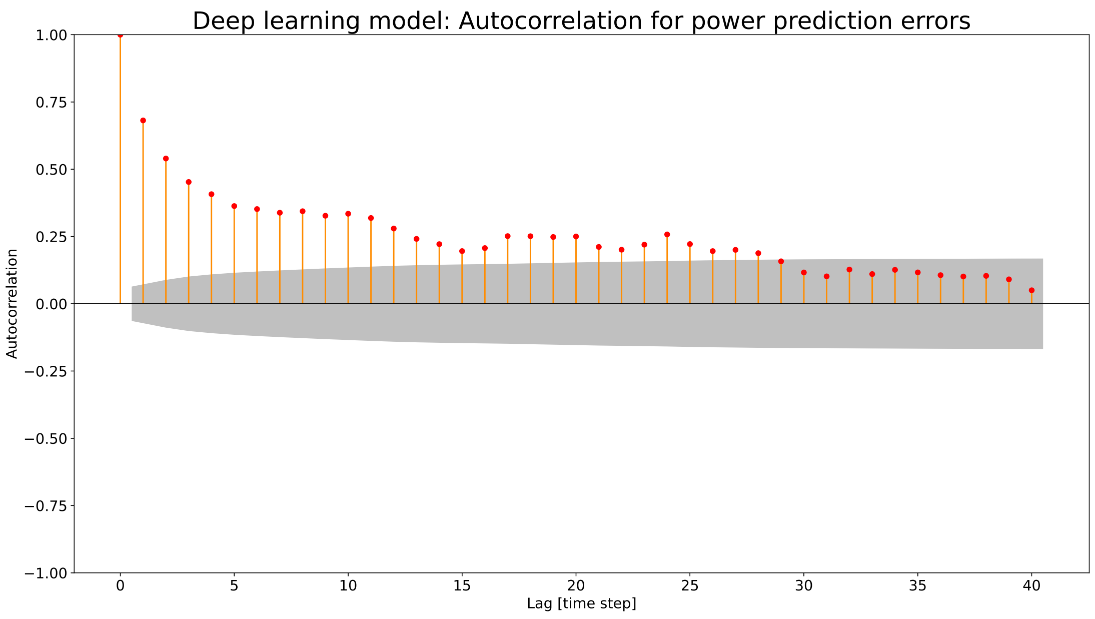

Figure 5.15: Autocorrelation for error time series from stochastic deep learning model trained on fifty weeks of data.

6 Discussion

The presented results originate from a little more than one month of test data in the spring. Evaluating the trained models on different sets of data may yield different results. The training data directly preceded the test set, with the shorter training sets only representing the winter and early spring. The result may be different if the model were trained or tested on other data sets of similar lengths. However, spring is a season with varying outside weather, and using that time to train and test models probably yielded more representative results than if the models were trained and tested on summer data. On the other side, winter is the season in Norway when heating is most needed and when the price of power consumption is at its highest, and evaluations of the models for that time could have been beneficial. As there are large differences in the seasons, different models for different conditions could be explored further. However, the results show that a large amount of data is needed for the convergence of model performance. Only having seasonal models would require years of data collection to achieve desired model performance.

6.1 Temperature Forecasting

The results in chapter 4 demonstrate the difficulty of creating optimal linear one-step-ahead forecasting models for temperature dynamics. Direct approaches are necessary to ensure robust forecasts. As expected, the performance of linear direct models improves with increasing data. The mean absolute errors for all forecast horizons seem to stabilize well under $1.0°C$ for models trained on more than 10 weeks of data. For some of the rooms, more data is needed, and for others, less. On the test set used, the forecasted temperature seems to be slightly biased and tends to forecast too high temperatures. This behavior is not easily explained.

The Temporal Fusion Transformers outperform the linear direct models on small training data sets. More than four weeks of data are needed to make the linear direct model outperform the TFT trained on one week of data. This makes the TFT a more viable model to use when little data has been collected. With fifty weeks of data available, the performance of the linear models is close to that of the TFT. The linear models have lower mean absolute errors for the early forecast horizons, but the TFT provides a better forecast 12 hours ahead for most of the rooms. Training the TFT takes substantially more resources than the linear models. This is not a problem for their uses since models can be trained offline, but its complex structure makes it more difficult to use in an MPC scheme. With similar performances when enough data is available, utilizing linear direct models in MPC is the best choice.

The linear model also manages to capture the thermal dynamics of the house. The results show that it is possible to achieve good models of linear households without deploying advanced simulation or mathematical modeling methods and that linear regression is a good basis for modeling the temperature dynamics of the household with enough data.

6.2 Power Prediction

The results of this thesis show that linear models are a better choice than feed-forward DNNs with the chosen architecture for predicting power consumption. The architecture consisted of units representing each heat pump. By analyzing the models it became clear that the deep learning model struggled with learning the weights for each unit. The linear models outperformed the deep learning ones when trained on short data sets, but showed similar performance when trained on fifty weeks of data. This may be a consequence of the structure chosen, and other deep learning-based methods could work better. Based on the model analysis and the prediction results on the test set, the linear models proves a better candidate to use in the MPC scheme than the deep learning ones.

On the chosen test data, the linear model trained on one week of data outperformed the models trained on more data. Thirty weeks of data were needed to achieve similar performance. Some of the linear models also struggled with a multitude of predictions close to zero for a wide range of true power consumption. This speaks to an unreliable model architecture that struggles to learn parameter weights. The data is dominated by zero-valued power consumption, and the models might overfit this data. Measures to limit these parts of the training data's contribution to updating the model weights might improve its performance. As stated earlier, it is more important that the power consumption models achieve consistent accuracy at all power consumption levels. For the trained models that have a high performance, the accuracy and uncertainty of the stochastic models are quite even for the whole range of common power consumption values. The test set consisted of few samples with power consumption values over 3 kWh, so the models' relative performance for that scenario is still unknown.

The power consumption models were constructed as static predictors that ignore possible temporal patterns in the data. There were some autocorrelations in the errors, which might indicate some temporal tendencies or that the data simply do not manage to capture all dynamics of the power consumption. Exploiting some temporal patterns might increase the performance of the model.

7 Conclusion

This Master's Thesis set out to explore pure machine learning-based modeling of indoor temperature and power consumption of heating in a household. Forecasting and prediction models for temperature and power consumption dynamics respectively. Both sets of models were created using linear regression and deep learning-based methods. These models were trained on data of varying lengths and evaluated using the same test set. Some analysis of the trained models was performed.

Stochastic direct linear regression-based temperature forecasters showed similar performance to the Temporal Fusion Transformer but struggled to match the deep learning method when trained on limited data. A comparison between direct and iterated linear models showed that the robustness of direct models is needed for this use case. The performance of the direct linear model increased with increasing training data size, and more than ten weeks of data are needed for the model's performance to stabilize. By analyzing the weights of past temperatures, it appears that the linear model manages to capture some temperature dynamics between the rooms in the house.

The linear models for predicting power consumption outperformed the deep learning variant which struggled to learn the dynamics of each heat pump unit. With increasing training set lengths, the mean absolute errors were quite stable, but the best performance came from models trained on one week and more than thirty weeks of data. An analysis of the autocorrelation of the model errors showed that the combination of data and model architecture does not manage to account for all power consumption dynamics. Other approaches should be explored.

7.1 Further Work

The scope of this thesis was to explore pure ML modeling methods. Improving or combining these with mathematical models to create hybrid models or using other modeling approaches such as Bayesian deep learning could be explored in future work. By using prior knowledge of the system to constrain parameters in ML-based models, they would reflect the true dynamics more accurately. One method is to regularize the weighting of past temperatures in the linear temperature forecasting model. Another method is to combine one-step-ahead forecasters with the direct forecaster for temperature forecasting.

Linear direct temperature models showed poor relative performance when trained on limited data, so the question of how to improve model performance with little data is still important. Improving the power consumption models is also important to work on in the future.

The proposed models were not implemented and evaluated in an MPC scheme during this Master's Thesis. This needs to be done to evaluate them in a closed-loop setting. The models were only trained and evaluated in one household. Training the models on data from several households would provide more insights into their overall performance.

Bibliography

[1] Bryan Lim, Sercan Ö. Arık, Nicolas Loeff, and Tomas Pfister. Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, October 2021. ISSN 01692070. doi: 10.1016/j.ijforecast.2021.03.012. URL https://linkinghub.elsevier.com/retrieve/pii/S0169207021000637.

[2] Magnus Blaker. Derfor er strømprisen høy, selv om magasinene er fulle, November 2022. URL https://www.nettavisen.no/5-95-783500.

[3] Slik blir strømstøtten for desember, . URL https://norgesenergi.no/stromsmart/statens-stromstotte-desember-2022/.

[4] Statnetts Kortsiktige markedsanalyse, . URL https://www.statnett.no/for-aktorer-i-kraftbransjen/planer-og-analyser/kortsiktig-markedsanalyse/.

[5] Slik har høye strømpriser påvirket husholdningenes økonomi, . URL https://www.ssb.no/energi-og-industri/energi/artikler/slik-har-hoye-strompriser-pavirket-husholdningenes-okonomi.

[6] Hva bruker mest strøm? | Strøm.no, . URL https://xn--strm-ira.no/hvor-mye-str%C3%B8m-bruker.

[7] Smart strømstyring for boliger | Søk om støtte. URL https://www.enova.no/privat/alle-energitiltak/smart-stromstyring/.

[8] A. Nacer, B. Marhic, and L. Delahoche. Smart Home, Smart HEMS, Smart heating: An overview of the latest products and trends. In *2017 6th International Conference on Systems and Control (ICSC)*, pages 90–95, Batna, Algeria, May 2017. IEEE. ISBN 9781509039609. doi: 10.1109/ICoSC.2017.7958713. URL http://ieeexplore.ieee.org/document/7958713/.

[9] Frauke Oldewurtel, Alessandra Parisio, Colin N. Jones, Dimitrios Gyalistras, Markus Gwerder, Vanessa Stauch, Beat Lehmann, and Manfred Morari. Use of model predictive control and weather forecasts for energy efficient building climate control. *Energy and Buildings*, 45:15–27, February 2012. ISSN 03787788. doi: 10.1016/j.enbuild.2011.09.022. URL https://linkinghub.elsevier.com/retrieve/pii/S0378778811004105.

[10] Welcome | TRNSYS : Transient System Simulation Tool. URL https://www.trnsys.com/.

[11] Silvia Di Già and Davide Papurello. Hybrid Models for Indoor Temperature Prediction Using Long Short Term Memory Networks—Case Study Energy Center. *Buildings*, 12 (7):933, July 2022. ISSN 2075-5309. doi: 10.3390/buildings12070933. URL https://www.mdpi.com/2075-5309/12/7/933.

[12] Raphael Linker and Ido Seginer. Greenhouse temperature modeling: a comparison between sigmoid neural networks and hybrid models. *Mathematics and Computers in Simulation*, 65(1-2):19–29, April 2004. ISSN 03784754. doi: 10.1016/j.matcom.2003.09.004. URL https://linkinghub.elsevier.com/retrieve/pii/S037847540300137X.

[13] Borui Cui, Cheng Fan, Jeffrey Munk, Ning Mao, Fu Xiao, Jin Dong, and Teja Kuruganti. A hybrid building thermal modeling approach for predicting temperatures in typical, detached, two-story houses. *Applied Energy*, 236:101–116, February 2019. ISSN 03062619. doi: 10.1016/j.apenergy.2018.11.077. URL https://linkinghub.elsevier.com/retrieve/pii/S0306261918317938.

[14] Rebecca Bevans. Multiple Linear Regression | A Quick Guide (Examples), February 2020. URL https://www.scribbr.com/statistics/multiple-linear-regression/.

[15] Steven Dye. Quantile Regression, February 2020. URL https://towardsdatascience.com/quantile-regression-ff2343c4a03.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

[17] Sebastian Ruder. An overview of gradient descent optimization algorithms, June 2017. URL http://arxiv.org/abs/1609.04747. arXiv:1609.04747 [cs].

[18] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989. ISSN 08936080. doi: 10.1016/0893-6080(89)90020-8. URL https://linkinghub.elsevier.com/retrieve/pii/0893608089900208.

[19] Alex Krizhevsky Ilya Sutskever Nitish Srivastava, Geoffrey Hinton and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(15):1929–1958, 2014.

[20] Massimiliano Marcellino, James H. Stock, and Mark W. Watson. A comparison of direct and iterated multistep AR methods for forecasting macroeconomic time series. *Journal of Econometrics*, 135(1-2):499–526, November 2006. ISSN 03044076. doi: 10.1016/j.jeconom.2005.07.020. URL https://linkinghub.elsevier.com/retrieve/pii/S030440760500165X.

[21] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, July 2020. ISSN 01692070. doi: 10.1016/j.ijforecast.

2019.07.001. URL https://linkinghub.elsevier.com/retrieve/pii/S0169207019301888.

[22] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/5cf68969fb67aa6082363a6d4e6468e2-Paper.pdf.

[23] Alexej Klushyn, Richard Kurle, Maximilian Soelch, Botond Cseke, and Patrick van der Smagt. Latent matters: Learning deep state-space models. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 10234–10245. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/54b2b21af94108d83c2a909d5b0a6a50-Paper.pdf.

[24] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A Multi-Horizon Quantile Recurrent Forecaster. *arXiv:1711.11053 [stat]*, June 2018. URL http://arxiv.org/abs/1711.11053. arXiv: 1711.11053.

[25] Florian A. Potra and Stephen J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1-2):281–302, December 2000. ISSN 03770427. doi: 10.1016/S0377-0427(00)00433-7. URL https://linkinghub.elsevier.com/retrieve/pii/S0377042700004337.

[26] Abdul Afram, Farrokh Janabi-Sharifi, Alan S. Fung, and Kaamran Raahemifar. Artificial neural network (ANN) based model predictive control (MPC) and optimization of HVAC systems: A state of the art review and case study of a residential HVAC system. *Energy and Buildings*, 141:96–113, April 2017. ISSN 0378-7788. doi: 10.1016/j.enbuild.2017.02.012. URL https://www.sciencedirect.com/science/article/pii/S0378778816310799.

[27] Boyang Zhang, Xiuxia Sun, Shuguang Liu, and Xiongfeng Deng. Recurrent Neural Network-Based Model Predictive Control for Multiple Unmanned Quadrotor Formation Flight. *International Journal of Aerospace Engineering*, 2019:1–18, June 2019. ISSN 1687-5966, 1687-5974. doi: 10.1155/2019/7272387. URL https://www.hindawi.com/journals/ijae/2019/7272387/.

[28] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpOASES: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, December 2014. ISSN 1867-2949, 1867-2957. doi: 10.1007/s12532-014-0071-1. URL http://link.springer.com/10.1007/s12532-014-0071-1.

[29] Sensibo: Smart Air Conditioner | Control Your AC With Your Phone, . URL https://sensibo.com/.

[30] Et uvanlig strømselskap Tibber. URL https://tibber.com/no.

[31] Martin Kværnes. Spår store svingninger i strømprisene fremover, October 2021. URL https://www.dn.no/energi/spar-store-svingninger-i-stromprisene-fremover/2-1-1083443.

[32] Julien Herzen, Francesco Lässig, Samuele Giuliano Piazzetta, Thomas Neuer, Léo Tafti, Guillaume Raille, Tomas Van Pottelbergh, Marek Pasieka, Andrzej Skrodzki, Nicolas Huguenin, Maxime Dumonal, Jan Kościsz, Dennis Bader, Frédérick Gusset, Mounir Benheddi, Camila Williamson, Michal Kosinski, Matej Petrik, and Gaël Grosch. Darts: User-friendly modern machine learning for time series, 2021.

Appendix

A The Temporal Fusion Transformer

This appendix is an extract from Section 3.3 of my Project Thesis with minor changes.

The Temporal Fusion transformer is an attention-based Deep Neural Network developed by Google. It is tested against other forecasting algorithms (such as DeepAR, ARIMA, and DSSM), and significantly outperforms them in terms of quantile losses on several real-world datasets [1]. The TFT is a direct method that performs multistep forecasting in contrast to iterated methods that use one-step-ahead forecasts in an iterative manner (such as DeepAR) where the errors may propagate for each feedback loop. The algorithm can train on multiple time series where the distribution of each series is unique. It has a local processing part that focuses on specific events in the data, and a global part that focuses on the combined characteristics of the whole dataset. It also supports several types of input data:

- Time-dependent data with known values in the past and future
- Time-dependent data only known up to forecasting time
- Static covariates that are time-invariant but characterize different time series used for training.

The architecture of the TFT is explained in the next section, and several aspects of the architecture make it a good choice for the task at hand. The TFT uses gating mechanisms to easily adapt to the complexity of the problem. If a linear mapping is enough, the gating mechanism will skip a more complex non-linear function mapping or an LSTM module during training resulting in a simpler model. This benefits the generalization of the model which makes it easier to use this model in new households without many manual changes to the network architecture. Some hyperparameters need to be tuned, but there exist automatic tuning procedures such as grid search or random search to adaptively find the best hyperparameters for the model of each household. The TFT also leverages the covariates better than most other models through the attention mechanism, and will therefore typically perform better on shorter datasets. This, in addition to its forecasting capabilities and support for a wide range of inputs, makes the TFT an intuitively good choice for modeling the temperature and power dynamics of a house.

A.1 Architecture

The TFT consists of several building blocks:

- Gating mechanisms are used to adaptively suppress part of the data that is not relevant. This gives the network an adaptive complexity to make it viable on a large variety of datasets.
- A variable selection network (VSN) is used to assign weights to the input variables to effectively select the most relevant input variables to the network
- Static Covaraiate Encoders are used to enable the use of static variables in the network. These can for example categorize different types of time series.

- Interpretable Multi-Head Attention is used to learn long-term patterns
- By simultaneously predicting outputs of various percentiles, the TFT provides quantile outputs.
- A quantile loss function is used to train the network

How each of these building blocks makes up the TFT is explained in the following sections. For consistency, the notation used in [1] is also used here. Figure A.1 gives an overview of the TFT architecture and the different building blocks are explained in the following sections

Gating mechanisms

Knowing the exact relations between the inputs and the target is often impossible. Because of this, it is difficult to assess whether we need a complex non-linear model or if it is enough with a simple model to process the inputs. Gated Linear Units (GLUs) can be used to suppress architecture that is not relevant to the given task. With an input $\gamma \in \mathbb{R}^{d_{model}}$ where d_{model} is the hidden state size (number of states in each layer that is shared throughout the network), the GLU is given by:

$$\mathrm{GLU}_{\omega}(\gamma) = \sigma\left(\boldsymbol{W}_{1,\omega}\gamma + \boldsymbol{b}_{1,\omega}\right) \cdot \left(\boldsymbol{W}_{1,\omega}\gamma + \boldsymbol{b}_{4,\omega}\right). \tag{1}$$

$\sigma(.)$ is the sigmoid activation function, $\boldsymbol{W}_{(.)} \in \mathbb{R}^{d_{model} \times d_{model}}, \boldsymbol{b}_{(.)} \in \mathbb{R}^{d_{model}}$ are the weights and biases, and $\cdot$ is the element-wise product. The subscript ω is an index used to denote weight sharing (for example across all time steps t). The GLU can then be used to determine what part of the input γ is passed on to the next layer.

The paper introduces the Gated Residual Network (GRN), a network that uses GLUs to adaptively use non-linear processing only when needed. The GRN takes an input $\boldsymbol{a}$ and optionally a context vector $\boldsymbol{c}$ (that provides a context for the input $\boldsymbol{a}$ as described by a static covariate) and outputs the original value in a linear fashion or with a nonlinear contribution. The GRN is given by:

$$\mathrm{GRN}_{\omega}(\boldsymbol{a}, \boldsymbol{c}) = \mathrm{LayerNorm}(\boldsymbol{a} + \mathrm{GLU}_{\omega}(\boldsymbol{\eta}_1)), \tag{2a}$$

$$\boldsymbol{\eta}_1 = \boldsymbol{W}_{2,\omega}\boldsymbol{\eta}_2 + \boldsymbol{b}_{2,\omega}, \tag{2b}$$

$$\boldsymbol{\eta}_2 = \mathrm{ELU}(\boldsymbol{W}_{3,\omega}\boldsymbol{a} + \boldsymbol{W}_{4,\omega}\boldsymbol{c} + \boldsymbol{b}_{3,\omega}). \tag{2c}$$

Here the ELU-operator is the Exponential Linear Unit activation function, GLU is as explained earlier and LayerNorm is layer normalization where the hidden units in the same layer are normalized. $\eta_{(.)}$ are intermediate values in the dense layers (and after the ELU-operation for η_2) as shown in figure A.2. In summary, the input to the GRN is first transformed by the ELU before a linear transformation and a possible dropout are performed. Finally, the GLU is applied to $\boldsymbol{\eta}_1$ and added to the original input to form a residual connection to possibly skip the layer before layer normalization is applied to produce the output.

Variable Selection Networks

Variable selection networks are used to select the input variables that are most relevant at each timestep. The variable selection can also filter out noisy inputs that may affect the performance. This is done for both static and time-dependent variables. The categorical variables cannot be processed by the VSN as is, so entity embeddings are used to map these variables into an

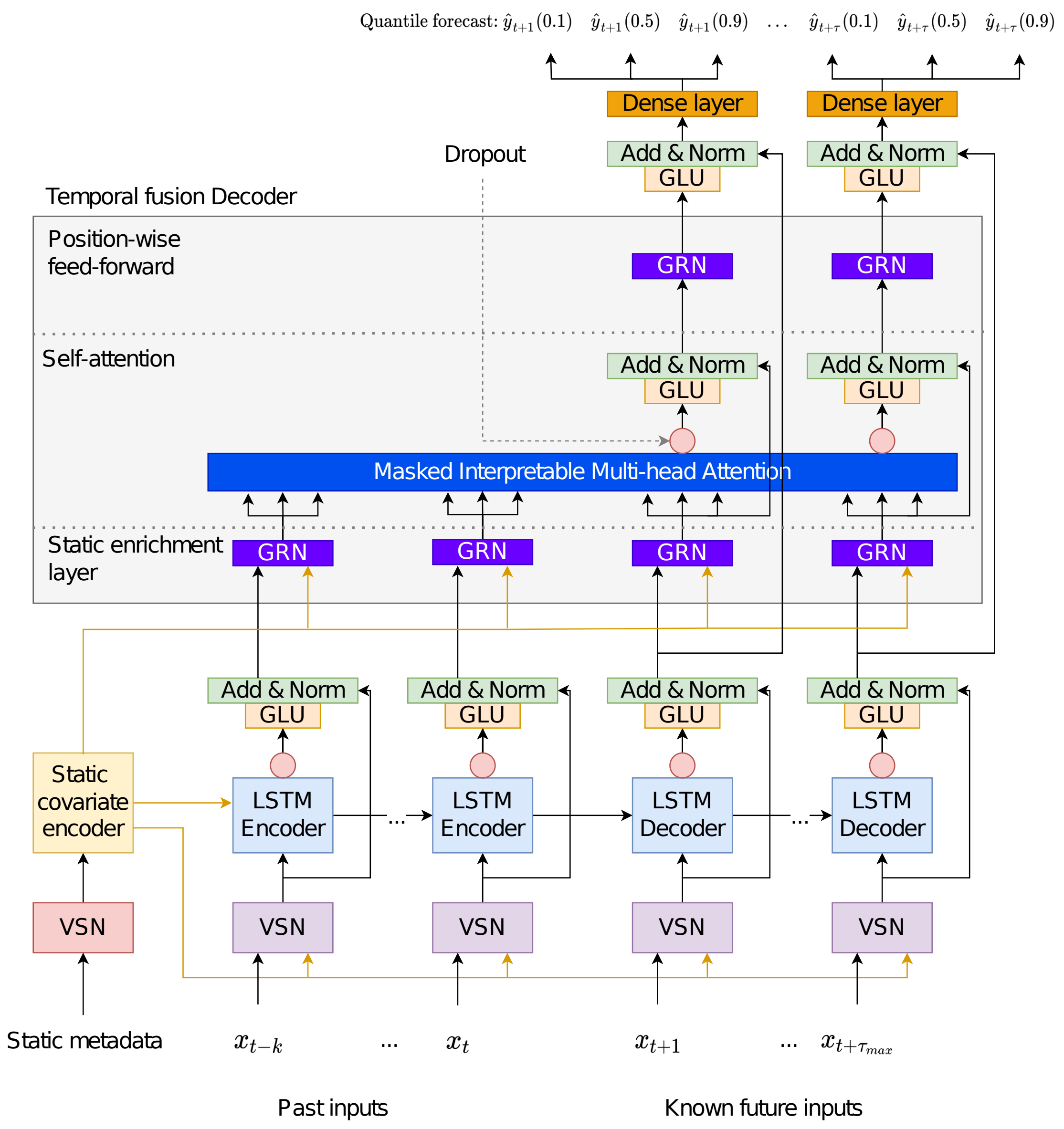

Figure A.1: The Temporal Fusion Transformer architecture from [1] as explained in section A.1.

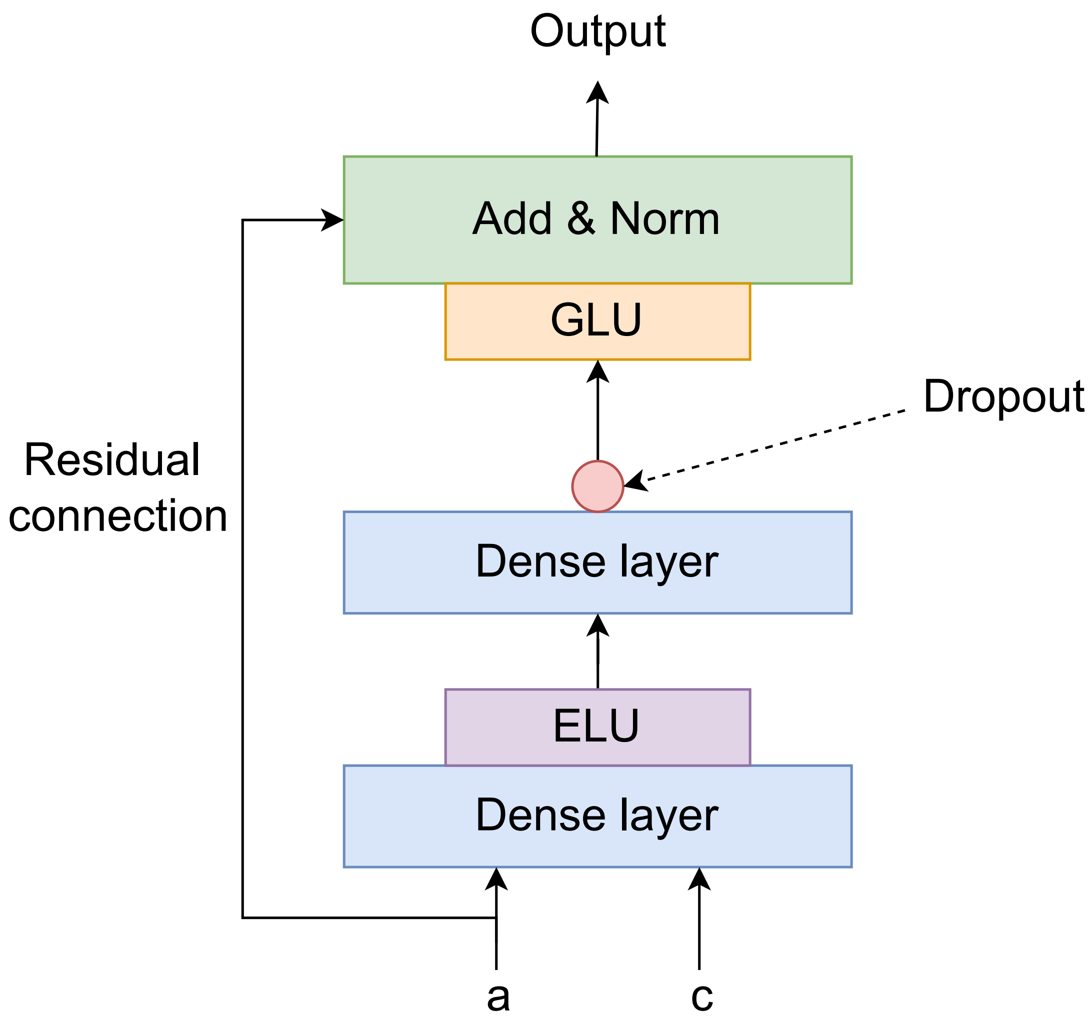

Figure A.2: Gated residual network architecture from [1].

d_{model}-dimensional vector. A linear transformation is applied to the continuous inputs x such that each input also is a d_{model}-dimensional vector. These transformations are done in a dense layer. In the TFT architecture, the static, past, and future inputs use separate VSNs, but they are all built the same way. The static metadata version of the VSN does not take any context vectors as inputs since it already contains contextual data.

We define $\boldsymbol{\xi}_t^{(j)} \in \mathbb{R}^{d_{model}}$ as the transformed input of the *j*-th variable at time *t*. $\Xi_t = \left[\boldsymbol{\xi}_t^{(1)^T}, ..., \boldsymbol{\xi}_t^{(m_x)^T}\right]^T$ is a vector with all inputs at time t. m_x is the total number of covariates (past, future or static). By sending Ξ_t and an external vector $\boldsymbol{c}_s$ (from the static covariate encoder) through a GRN and then into a Softmax activation function, variable selection weights, $\boldsymbol{v}_{x_t}$, are obtained. This is shown in equation (3).

$$\boldsymbol{v}_{x_t} = \text{Softmax}\left(\text{GRN}_{v_x}(\Xi_t, \boldsymbol{c}_s)\right), \tag{3}$$

For static variables, $\boldsymbol{c}_s$ is omitted. In a separate layer, each $\boldsymbol{\xi}_t^{(j)}$ is fed through its own GRN:

$$\tilde{\boldsymbol{\xi}}_t^{(j)} = \text{GRN}_{\tilde{\xi}(j)}\left(\boldsymbol{\xi}_t^{(j)}\right). \tag{4}$$

I.e, each variable has its own GRN, and the weights are shared across all time steps t. Finally, $\tilde{\xi}_t^{(j)}$ is weighted with $\boldsymbol{v}_{xt}^{(j)}$, the j-th element of $\boldsymbol{v}_{x_t}$ to obtain $\tilde{\xi}_t$, the variable selection output:

$$\tilde{\boldsymbol{\xi}}_t = \sum_{j=1}^{m_x} \boldsymbol{v}_{x_t}^{(j)} \tilde{\boldsymbol{\xi}}_t^{(j)}, \tag{5}$$

where each input is weighted to signify its relevance. This architecture is displayed in figure A.3.

Static Covariate Encoders

GRN encoders are used to obtain four different context vectors to be used in various places in the TFT architecture. These vectors contain information from static metadata such as categorization for several time series used in the model. The four context vectors are:

- $\boldsymbol{c}_s$ is used in temporal variable selection.
- $\boldsymbol{c}_c$ is used in the local processing of temporal features.
- $\boldsymbol{c}_h$ is used in the local processing of temporal features.
- $\boldsymbol{c}_e$ is used to enrich temporal features with static information.

The static metadata is fed through a VSN to generate $\boldsymbol{\zeta}$ which is the input to the Static Covariate Encoder. To obtain the context vectors, $\boldsymbol{\zeta}$ is encoded by $\boldsymbol{c}_n = \text{GRN}_{c_n}(\boldsymbol{\zeta})$ where $n \in \{s, c, h, e\}$.

Interpretable Multi-Head Attention

Transformer-based architectures utilize multi-head attention to learn long-term relationships in the data. The general, the attention mechanism consists of three components; values $\boldsymbol{V} \in \mathbb{R}^{N \times d_V}$, keys $\boldsymbol{K} \in \mathbb{R}^{N \times d_{attn}}$ and queries $\boldsymbol{Q} \in \mathbb{R}^{N \times d_{attn}}$ where N is the number of time steps and d_v and d_{attn} being the number of hidden states in the value-vectors, and keys- and queries-vector respectively. The method takes the query vector attributed to a timestep value in

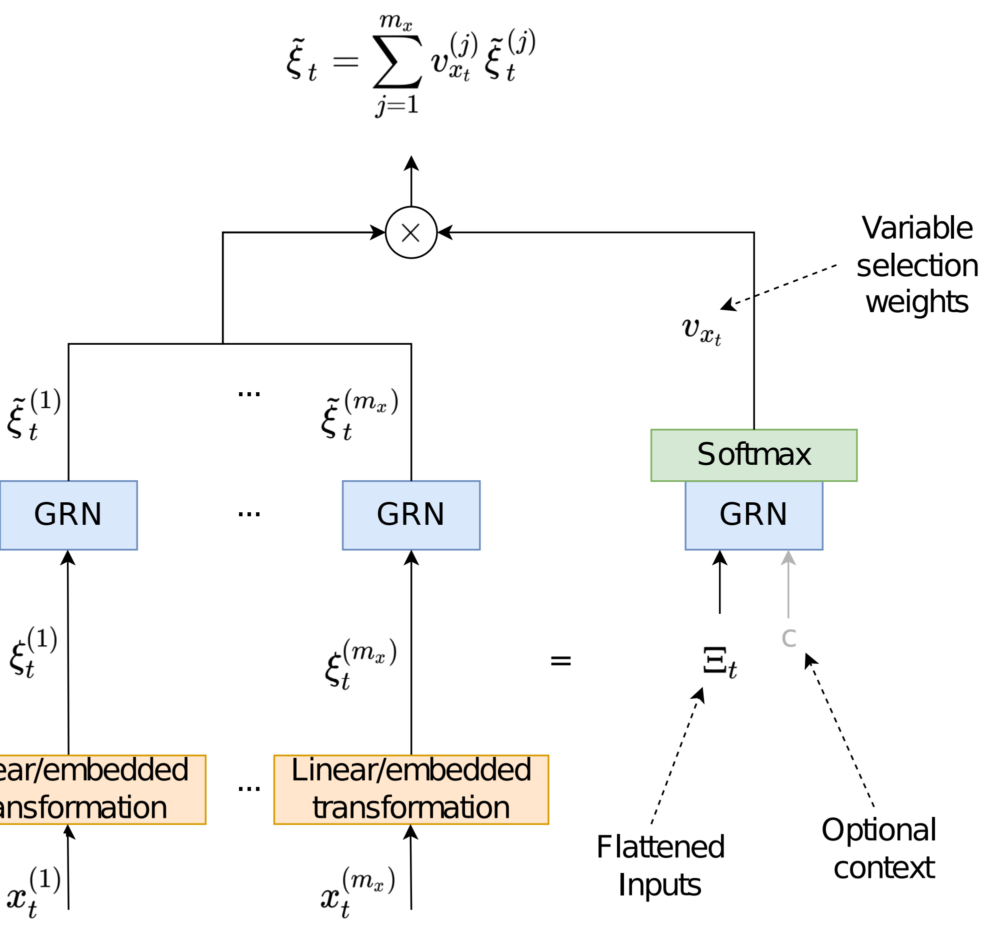

Figure A.3: Variable selection network architecture from [1].

the time series and creates weights for all values by scoring it against each key in the database. This operation gives information about how the value relates to others in the time series. It is common to use the Softmax activation function with scaled dot-product to compute these weights:

$$A(\boldsymbol{Q},\boldsymbol{K}) = \text{Softmax}\left(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d_{attn}}\right). \tag{6}$$

Then, the attention method scales the values in the time series by:

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = A(\boldsymbol{Q},\boldsymbol{K})\boldsymbol{V}. \tag{7}$$

This method can be improved to learn different representation subspaces by introducing multi-head attention:

$$\text{MultiHead}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = [\boldsymbol{H}_1, ..., \boldsymbol{H}_{m_H}]\boldsymbol{W}_H, \tag{8}$$

$$\boldsymbol{H}_h = \text{Attention}\left(\boldsymbol{Q}\boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\boldsymbol{W}_K^{(h)}, \boldsymbol{V}\boldsymbol{W}_V^{(h)}\right), \tag{9}$$

where $\boldsymbol{H}_h$ is attention head $h \in [1, m_H]$ and m_h is the number of heads. $\boldsymbol{W}_K^{(h)} \in \mathbb{R}^{d_{model} \times d_{attn}}$, $\boldsymbol{W}_Q^{(h)} \in \mathbb{R}^{d_{model} \times d_{attn}}$, $\boldsymbol{W}_V^{(h)} \in \mathbb{R}^{d_{model} \times d_V}$ are weights for each head h for the keys, queries and values. $\boldsymbol{W}_H \in \mathbb{R}^{(m_H \cdot d_V) \times d_{model}}$ combines the output for each head linearly.

This method uses different values in each head, and as such, the attention weights give no information about a feature's importance. The multi-head attention method above can be changed to make the heads share values while learning different types of temporal relationships:

$$\text{InterpretableMultiHead}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \tilde{\boldsymbol{H}}\boldsymbol{W}_H \tag{10}$$

$$\tilde{\boldsymbol{H}} = \tilde{A}(\boldsymbol{Q},\boldsymbol{K})\boldsymbol{V}\boldsymbol{W}_V, \tag{11a}$$

$$= \left\{1/H \sum_{h=1}^{m_H} A\left(\boldsymbol{Q}\boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\boldsymbol{W}_K^{(h)}\right)\right\}\boldsymbol{V}\boldsymbol{W}_V, \tag{11b}$$

$$= 1/H \sum_{h=1}^{m_H} \text{Attention}\left(\boldsymbol{Q}\boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\boldsymbol{W}_K^{(h)}, \boldsymbol{V}\boldsymbol{W}_V\right), \tag{11c}$$

where $\boldsymbol{W}_V \in \mathbb{R}^{d_{model} \times d_V}$ are value weights for all heads and$\boldsymbol{W}_H \in \mathbb{R}^{d_{attn} \times d_{model}}$ is the final linear mapping. H is used to calculate the mean of the heads for the output.

Temporal Fusion Decoder

As shown in figure A.1, the input is initially passed through VSNs to encode and weigh the covariates. Before we perform multi-head attention on the data, the outputs of the VSNs are sent through a sequence-to-sequence model to make sense of the sequential ordering. An LSTM Encoder-Decoder model is used in this layer. The past inputs $\tilde{\boldsymbol{\xi}}_{t-k:t}$ is fed into the encoder while the future inputs $\tilde{\boldsymbol{\xi}}_{t+1:t+\tau_{max}}$ is fed into the decoder to produce context-aware embeddings that serve as the input to the Temporal Fusion Decoder. To account for the static metadata, the first LSTM cell state, and hidden state are initialized with the $\boldsymbol{c}_c$ and $\boldsymbol{c}_h$ context vectors respectively. Finally, a gated skip connection is used to possibly skip the layer if needed:

$$\tilde{\boldsymbol{\phi}}(t,n) = \text{LayerNorm}\left(\tilde{\boldsymbol{\xi}}_{t+n} + \text{GLU}_{\tilde{\phi}}(\boldsymbol{\phi}(t,n))\right), \tag{12}$$

where $\boldsymbol{\phi}(t,n) \in \{\boldsymbol{\phi}(t,-k),...,\boldsymbol{\phi}(t,\tau_{max})\}$ is the output from the LSTM encoder decoder and $n \in [-k, \tau_{max}]$ is a position/time index.

The first part of the Temporal Fusion Decoder is the *Static Enrichment Layer*. This layer uses the context vector $\boldsymbol{c}_e$ from the Static Covariate Encoder to enhance the temporal feature using a GRN to account for the static metadata:

$$\boldsymbol{\theta}(t,n) = \text{GRN}_\theta\left(\tilde{\boldsymbol{\phi}}(t,n), \boldsymbol{c}_e\right), \tag{13}$$

where n is the position index. The weights in the GRN$_\theta$ are shared across the entire layer.

Then self-attention is applied using the interpretable multi-head attention method described earlier. We collect all the temporal features into the matrix $\boldsymbol{\Theta}(t) = [\boldsymbol{\theta}(t,-k),...,\boldsymbol{\theta}(t,\tau)]^T$ and perform interpretable multi-head attention at each forecast timestep with $\boldsymbol{\Theta}(t)$ being the values, keys, and queries in the method:

$$\boldsymbol{B}(t) = \text{InterpretableMultiHead}(\boldsymbol{\Theta}(t), \boldsymbol{\Theta}(t), \boldsymbol{\Theta}(t)), \tag{14}$$

with the following values for the parameters in the model:

$$N = \tau_{max} + k + 1, \tag{15a}$$

$$d_V = d_{attn} = d_{model}/m_H, \tag{15b}$$

where m_H is the number of heads. This results in the output matrix $\boldsymbol{B}(t) = [\boldsymbol{\beta}(t,-k),...,\boldsymbol{\beta}(t,\tau)]$ To ensure that the multi-head attention is not "cheating" by looking at future values in the time series, decoder masking is applied to the layer. This method preserves the causal information flow by setting future values to "-inf" before the Softmax step in the multi-head attention method. A gating layer is also applied here in case using a simple method is enough during training:

$$\boldsymbol{\delta}(t,n) = \text{LayerNorm}\left(\boldsymbol{\theta}(t,n) + \text{GLU}_\delta(\boldsymbol{\beta}(t,n))\right). \tag{16}$$

The last part of the Temporal Fusion Decoder consists of a position-wise feed-forward Layer. This layer uses the GRN to apply non-linear processing to the output of the multi-head attention layer:

$$\boldsymbol{\psi}(t,n) = \text{GRN}_\psi\left(\boldsymbol{\delta}(t,n).\boldsymbol{c}_e\right). \tag{17}$$

As usual, the weights are shared across the layer. Finally, in order to adaptively skip the whole Temporal Fusion Decoder if the complexity is not needed, a gating layer is applied that connects the output from the sequence-to-sequence layer with the output from the Temporal Fusion Decoder:

$$\tilde{\boldsymbol{\psi}}(t,n) = \text{LayerNorm}\left(\tilde{\boldsymbol{\phi}}(t,n) + \text{GLU}_{\tilde{\psi}}(\boldsymbol{\psi}(t,n))\right), \tag{18}$$

Quantile outputs

The TFT provides prediction intervals by simultaneously predicting outputs of various percentiles at each time step. This can for example be $\{0.1, 0.5, 0.9\}$-quantiles. The output of quantile 0.9 will for example over-predict 90% of the time while the 0.5-quantile provides the mean forecast. These quantile forecasts are produced by using a quantile linear transformation learned from the quantile loss function (as described in the next section):

$$\tilde{y}(q,t,\tau) = \boldsymbol{W}_q\tilde{\boldsymbol{\psi}}(t,\tau) + b_q, \tag{19}$$

where $\tau \in \{1, ..., \tau_{max}\}$ are forecast time steps into the future and $\boldsymbol{W}_q \in \mathbb{R}^{1\times d}$ and $b_q \in \mathbb{R}$ are linear coefficients for quantile q with d being the length of $\tilde{\boldsymbol{\psi}}(t, \tau)$.

Quantile loss function

The TFT utilized a quantile loss function for training:

$$\mathcal{L}(\Omega, \boldsymbol{W}) = \sum_{y_t\in\Omega} \sum_{q\in\mathcal{Q}} \sum_{\tau=1}^{\tau_{max}} \frac{QL(y_t, \hat{y}(q, t-\tau, \tau), q)}{M_{\tau_{max}}}, \tag{20}$$

$$QL(y_t, \tilde{y}, q) = q(y - \hat{y})_+ + (1-q)(\hat{y} - y)_+. \tag{21}$$

Here $\boldsymbol{W}$ is the collection of all weights in the TFT model, Ω is the whole training set with M samples, and $\mathcal{Q}$ is the set of quantiles. This can for example be $\mathcal{Q} = \{0.1, 0.5, 0.9\}$. $(.)_+$ is the $\max(0, .)$-operator. The training consists of learning the network weights $\boldsymbol{W}$ by minimizing this quantile loss function using the average over all the quantiles. This optimization can be performed using stochastic gradient descent, or more popularly, the Adam optimization algorithm [16].

A.2 Hyperparameters

There are several hyperparameters to be set in the model. Some are based on the requirements of the model, but most should be tuned to increase performance. Here are the hyperparameters and how they affect the training of the model:

- **Output length**: This is the decoder length in the model that determines the forecasting length. This value is set based on how long we want the model to forecast.
- **Input length**: This is the encoder length that determines how many past timesteps are used in the forecast. This value should be significantly larger than the output length, but using too many past values will increase the training time drastically.
- **Hidden state size**: This is the number of hidden states at each time step that is common across the whole architecture. It represents the capacity of the model where a larger hidden size allows the model to learn more complex patterns in the data. It is used in the GRN and LSTM layers.
- **LSTM layers**: Number of layers for the LSTM encoder and decoder. Normally using 1 layer is a good default.
- **Epochs**: The number of epochs to train the model (i.e. for how long we wish to train the model). The performance of the model on the training set will increase with the number of epochs, but after a certain point, the model will be overfitted, and decrease its generalization. This parameter can be tuned by monitoring the loss function or using the early stopping method.
- **Attention heads**: Number of attention heads to use.
- **Dropout rate**: Chance of "dropping" the input to a gated layer. Using dropout is a regularization technique to avoid overfitting [19].
- **Batch size**: number of series to send through the network for each training loop. Having a small batch size requires less memory, and the network trains faster. However, with small batch size, the accuracy of the gradient of the loss function may decrease.

- **Number of samples** the forecast draws from the probabilistic model during forecasting.