UPPSALA
UNIVERSITET

UPTEC F 22041
Examensarbete 30 hp
June 2022

Investigation of Machine Learning Regression Techniques to Predict Critical Heat Flux

Emil Helmryd Grosfilley

UPPSALA
UNIVERSITET

Investigation of Machine Learning Regression Techniques to Predict Critical Heat Flux

Emil Helmryd Grosfilley

Abstract

A unifying model for Critical Heat Flux (CHF) prediction has been elusive for over 60 years. With the release of the data utilized in the making of the 2006 Groeneveld Lookup table (LUT), by far the largest public CHF database available to date, data-driven predictions on a large variable space can be performed. The popularization of machine learning techniques to solve regression problems allows for deeper and more advanced tools when analyzing the data. We compare three different machine learning algorithms to predict the occurrence of CHF in vertical, uniformly heated round tubes. For each selected algorithm (v-Support vector regression, Gaussian process regression, and Neural network regression), an optimized hyperparameter set is fitted. The best performing algorithm is the Neural network, which achieves a standard deviation of the prediction/measured factor three times lower than the LUT, while the Gaussian process regression and the v-Support vector regression both lead to two times lower standard deviation. All algorithms significantly outperform the LUT prediction performance. The neural network model and training methodology are designed to prevent overfitting, which is confirmed by data analysis of the predictions. Additionally, a feasibility study of transfer learning and uncertainty quantification is performed, to investigate potential future applications.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala/Visby

Handledare: Jean-Marie LeCorre Ämnesgranskare: Alexander Medvedev

Examinator: Tomas Nyberg

Popolärvetenskaplig sammanfattning

Critical heat flux, som kan översättas till kritiskt värmeflöde, är ett farligt fenomen som kan förekomma i en reaktor i ett kärnkraftverk. Kritiskt värmeflöde innebär att kärnbränslet inte längre kan kylas ner och överhettar. Överhettning av ett kärnkraftverk måste undvikas, och man måste då ha en modell för när fenomenet kan förekomma, så att man kan säkerställa att det inte inträffar.

Här används datadrivna modeller för att förutspå när kritiskt värmeflöde kommer inträffa. Datadrivna modeller är modeller som skapats genom att anpassa modellen till fysiskt framtagen data. Datadrivna modeller är mycket användbara i fall där analytiska modeller inte kan framställas, alltså där bakomliggande fysiken ännu ej är helt förstådd. För att kunna framställa välfungerande datadrivna modeller krävs stora mängder data. Publikationen av en ny, stor databas skapar nya möjligheter att använda mer komplexa datadrivna algoritmer för att bygga modeller.

Databasen som användes för framtagandet av modellerna innehåller cirka 25000 datapunkter som alla innehåller trycket, massflödet och ångkvaliten vid tidspunkten där kritiskt värmeflöde förekom under tester. Dessutom finns geometrisk data som testerna genomfördes i, nämligen rörens längd och bredd. Dessa fem parametrar kan användas som indata i modellerna, och användas för att förutspå när kritiskt värmeflöde skulle förekomma med denna parameter kombination.

Här har tre olika algoritmer använts för att bygga upp modeller. ν-Support vektor regression, Gaussisk process regression och Neruala nätvärk. Dessa tre algortimer använder den stora databasen för att lära sig mönster som finns i datan, och bygga en model som kan återskapa datan. Modellerna kan sedan användas på data där man inte vet när kritiskt värmeflöde kommer att inträffa, och ge en prediktion på när det kan komma att ske.

Varje algoritm optimiserades för att framställa den bästa möjliga modellen för varje algoritm. Det neurala nätvärket producerade klart bättre modeller än de andra två algoritmerna. Alla optimiserade modeller gav klart bättre resultat än Groenevelds uppslagstabbell som används idag, och som gjorts på samma databas som våra modeller.

Då modellen som skapats med neurala nätvärk är klart bäst, gjordes fördjupade analyser på denna modell. Överanpassning av modellen till datan som används för att bygga modellen är ett välkännt problem för neurala nätverk. Flera tester gjordes därför för att säkerställa att åtgärderna vidtagna är nog för att modellen inte ska vara överanpassad. Modellens beteende liknar beteendet från uppslagstabellen, men är något mer komplext. Dessutom används en extra storlek, längden, i våran modell, vilket leder till en bättre anpassad modell, då den får mer information.

Slutligen gjordes försök till ”Transfer learning”, vilket går utt på att överföra mönstrena modellen har lärt sig, till ett liknande problem med lite annorlunda fysik. Ett exempel på detta är att överföra den framtagna modellen som bygger på data där vatten används som kylningsmedel, till data där någon annan vätska används som kylningsmedel. Eftersom kylningsmedlet påvärkar när kritiskt värmeflöde händer, måste detta ta hänsyn till. Att skapa en ny databas är mycket kostsammt och därför skulle möjligheten att överföra våran modell vara mycket lönsamm. Testerna som utfördes tyder på att detta är möjligt men inte perfekt för vissa typer av överförningar. Mer forskning på detta område är nödvändigt för att kunna fastställa om ”Transfer learning” är applicerbart.

Abbreviations and Definitions

Definitions

Cross Validation: A method used to test model predictions where multiple resamplings, or folds, are used to improve the generalization of the predictions tested.

Epochs: One epoch is when all the data has passed through the network once during training.

Exploitation: Using what we already know about our objective function to improve our guesses.

Exploration: Exploring parts of our objective function of which we have little knowledge, in the hopes that it will lead to better guesses.

Generalization: A generalized model will have learned the underlying physics, which allows it to generalize to data it has not seen previously.

Hyper-parameters: parameters that are used to control the learning process.

Objective function: a function to minimize the loss by tweaking hyperparameters.

Test set : The dataset used to test the performance of our models.

Train set : The dataset used to train our model.

Validation set : The dataset used in NN to measure the error during training.

Abbreviations

ADAM : Adaptive moment estimation
BO : Boiling number
CV : Cross validation
CHF : Critical Heat Flux [kW /m^2]
D : Diameter of the tube [m]
DH_{in} : = Inlet subcooling [J /Kg]
G : Mass Flux [kg/m^2s]
$\mathcal{GP}$ **:** Gaussian Processes
GPR : Gaussian Process Regression
LF : Loss Function
LML : Log-marginal-likelyhood
LUT : Lookup table (the 2006 Groeneveld lookup table)
M/P : Measured CHF / Predicted CHF
MSE : Mean Squared Error
MSLE : Mean Squared Logarithmic Error
NRMSE : Normalized Root Mean Squared Error
NN : Neural Network
P : Pressure [Pa]
ReLu : Rectifying Linear Unit
SGD : Stochastic Gradient Decent
Std : Standard deviation of the M/P term
SVM : Support Vector Machines
USNRC : United States Nuclear Regulatory Comission
We : Weber number
X : Quality (of vapor)
\ **:** Left matrix divide: $A\backslash b$ is the vector x which solves Ax = b
ε**-SVR :** ε-Support Vector Regression
ν**-SVR :** ν-Support Vector Regression
ρ_f **:** Fluid density [Kg/m^3]
ρ_g **:** Gas density [Kg/m^3]
ρ_r **:** Density ratio ρ_f / ρ_g

Contents

1 Introduction

Critical Heat Flux (CHF), is a complex physical phenomenon that occurs when heat-transfer capabilities between nuclear fuel and a cooling agent rapidly decrease. This can result in serious damage to the reactor and needs to be prevented. The simplest way of preventing CHF is by predicting when it will occur and staying clear of that parameter combination. However, predicting when CHF will occur is a complex task, and a unifying model or theory has been elusive for over 60 years. Many models have been proposed, both theoretical and data-driven models with varying levels of success.

Data-driven models have existed for a long time, however, in recent years this approach has taken up more of the research into CHF. The advancements in computational power as well as in model building, which have improved continuously since their conception, open up new possibilities. This has led to the popularization of machine learning algorithms in the field of CHF prediction, and has opened up new opportunities to improve models without improving the data the model is based on.

Many articles have investigated the possibilities of using machine learning to improve CHF predictions, these will be discussed in Section 2.2. Generally, we can say that the machine learning models seem to capture more information from the data than in previous research. However, they all utilize small databases with limited parameter spaces. The release of the largest CHF occurrence database to date by the United States Nuclear Regulatory Commission (USNRC) allows us to attempt a more general model.

Building on the aforementioned research into the topic we have decided to compare three promising model-building algorithms: ν-Support vector regression, Gaussian process regression, and Neural network regression. Combining these highly performing algorithms with a previously unavailable database we hope to build models that can significantly outperform previous general models.

One large drawback of data-driven models is that they are hard to apply to other physics than where the data comes from. In our case, the data comes from uniformly heated round tubes, while an industrial boiling system usually consists of more complex geometries, such as rod bundles for nuclear reactors. We will thus also look into the usage of transfer learning, which could help utilize our large database to learn the underlying physics and transfer the knowledge to different geometries, requiring only smaller databases in the new geometries.

2 Background

2.1 Critical Heat Flux

In an industrial boiling system, CHF is a thermal limit that defines the breaking point at which the heat transfer between the heating surface and the coolant no longer increases when the imposed heat flux through the heating surface increases. The heating surface can no longer transfer sufficient heat and rapidly rises in temperature while changing the heat transfer mechanism.

If CHF is reached, the boiling system needs to interrupt operations and be allowed to cool down for damage inspection, which leads to a loss in production. In extreme cases, CHF can also lead to serious damage to the boiling system, such as physical burnout, and in the case of a nuclear reactor, fuel failure. It is therefore imperative to not reach the CHF limit during operation. This is where the need for accurate predictions becomes important.

CHF can occur in different ways, which is one of the main reasons why a unifying model for all CHF predictions has yet to be found, despite over 60 years of research into the topic. The many types of CHF all lead to the same phenomenon but might occur due to different reasons.

To give the reader an idea of the range of CHF models that have been proposed previously, we can look at a paper that gives an overview of investigations done in the last 40 years [1]. The following mechanisms have been proposed to describe the occurrence of CHF: homogeneous nucleation, single nucleation, liquid sublayer dryout, bubble crowding, interfacial lift-off, boundary layer separation, hydrodynamic limit, liquid layer superheat limit, and various dryout film models.

Currently, most empirical CHF prediction models use analytical functions that depend on pressure (P), mass flux (G), diameter (D), and quality (X) [2]:

$$CHF = f(P, G, D, X) \tag{1}$$

As we also have the length (L) as a parameter in our data set, we will also look at the influence of this parameter on our predictions. Using Dimensional analysis [2], we can reduce the number of parameters to three, while theoretically keeping the same information. This is done by combining our existent variables so that they become dimensionless. We utilize the boiling number (BO) at the occurrence of CHF, the ratio between the density of the gas (ρ_g) and density of the fluid (ρ_f), and the Weber number (We):

$$BO_{CHF} = f(\rho_r, X, We) \tag{2}$$

Using dimensionless parameters has the added benefit that parameters are easier to translate. For example from one geometry to another, compared to a diameter parameter that has no direct translation in, for example, a fuel bundle. Another example is translating the model from one coolant fluid (e.g. water) to another.

2.2 Machine Learning to Predict Critical Heat Flux

Models produced with a data-driven approach have for a long time been used to predict the occurrence of CHF [1]. However, these have often used simpler correlations such as polynomials [3], or other analytical functions. With the growth of computational power, a new approach to data-driven predictions has received more attention in the field of nuclear research[4]: utilizing machine learning to design prediction models, resulting in more sophisticated models than previously.

The machine learning algorithms used for predicting CHF are usually supervised regression algorithms, meaning that they use labeled data while training the model and attempt to predict the value of CHF. The most common types of machine learning algorithms used in previous research consulted for this thesis have been: fully connected neural networks [5][6], mixes of other neural network types [7][8][9], ε-Support vector regression[5], ν-Support vector regression [8] [10], Gaussian process regression [11], Parametric Gaussian process regression [12], Decision trees [5], Random forests [5][6], and Least-squares regression [5].

Directly comparing the different models is a complicated task, as they all use different ways to report their results. We can nonetheless draw some conclusions from the different types of models. The Decision trees and Random forests tend to overfit the data. ν-Support vector regression outperforms ε-Support vector regression. The mixes of different types of neural networks are harder to generalize and more challenging to implement than the fully connected neural networks.

One large drawback with these models is the data they are based on, as a machine learning model can only be as good as its training data allows it. Many published models are either based on a single database or model-generated data. Single databases rarely exceed 1000 data points, and can thus not pretend to capture the underlying physics to CHF in more than a small parameter space. Model-generated data is data that is based on an underlying model, that generates data with random deviations. If advanced ML tools are used on such data the resulting model would predict the underlying data-generating model, not the actual physics of CHF. To improve these predictions, utilizing a larger database from actual experimental data is therefore paramount.

2.3 Lookup Table

A standard model to roughly predicts the margins of CHF in a nuclear reactor is the 2006 Groeneveld Lookup table (LUT) [3]. The LUT can be used to predict CHF in a large variable space and is a data-driven approximation of the CHF phenomenon. It can be described as "a normalized data bank for vertical 8 mm water-cooled tube" [3]. We will use the LUT as a reference to compare the results of our prediction methods.

The data that is used to construct the LUT is roughly the same data that is described in Section 3.1, and that will be used to build our models. In addition, the LUT utilizes some proprietary

data that were not released. To have the model in table form the number of variables used to predict CHF needs to be limited to three for readability. Since they are generally seen as the most influential parameters, the LUT utilizes the pressure, mass flux, and quality as parameters. In addition, a correction term for the diameter of the tube is defined as [3]:

$$CHF = CHF_{D_{8mm}} \left(\frac{D_{real}}{8} \right)^{-1/2} \tag{3}$$

Where $CHF_{D_{8mm}}$ is the CHF that is given by the lookup table, which is normalized to 8mm, and D_{real} is the real diameter in millimeters. When reading $CHF_{D_{8mm}}$ from the LUT we also need to interpolate, since the table only contains a limited set of data points, this is done with a linear interpolator. The length of the tube was not used in the approximation, since it was seen as a second-order parameter by the author, for sufficiently large $\frac{L}{D}$.

Table 1: The parameters spanned by the LUT

Parameter	minimum	step-size	maximum
P [kPa]	100	200 - 3000	21000
G [kg/m^{2}s]	0	200 - 500	8000
X	-0.5	0.05 - 0.1	1

The LUT is created using 1st-degree polynomials to approximate data, coupled with extrapolation and/or previous models where there is no data. The irregularities produced are then smoothed. It is reported to have a root mean squared error of 38.92% when using local quality, and 7.10% when using inlet parameters [3]. A notable difference with our models is that the LUT utilizes all the data in building the model. Doing this when using ML is not feasible, since we also need to test our models for generalization. Our algorithms will thus use less data than the LUT when building the models.

3 Data Description

3.1 Database

The data used in this thesis has been collected by Groeneveld to produce the LUT described in Section 2.3. The data was recently published by the USNRC [13].

The data contains roughly 25000 measurements and comes from 62 different sources. The data was collected during the last 60 years. The methods used to measure the CHF are diverse. The methods used for CHF identification are: visual identification, physical burnout, changes in the test section resistances, and the usage of thermocouples. Because of the wide variety of measurement types and the long time span, which has seen new techniques employed, the uncertainty of the data-set is consequential. The uncertainty is also inconsistent over data points since they have different identification methods. There is no uncertainty estimation given with the data.

Additionally, multiple parameters, seen as secondary, are not part of the data set, which can lead to unmeasured differences between data sets. Groeneveld provides a list of such parameters that are considered secondary: test-section orientation, test-section material, type of heating, wall thickness, surface roughness, inlet/outlet throttling, and dissolved gas content in the coolant [13]. At least some of these unaccounted variables lead to extra uncertainties in the data.

The ranges covered by the data are quite substantial, as seen in Table 2, compared to what previous models use, but are not equally distributed over the whole span, as seen in Figure 1. We see that none of the variables have even distributions, which will inevitably lead to the model performing better in some regions than others.

Table 2: The parameter-span of the dataset

Variable	CHF [kW/m^2]	P [kPa]	G [kg/m^{2}s]	X	D [m]	L [m]
Min Value	50	100	8,2	-0,497	0,002	0,05
Max Value	16339,3	20000	7964	0,999	0,016	20

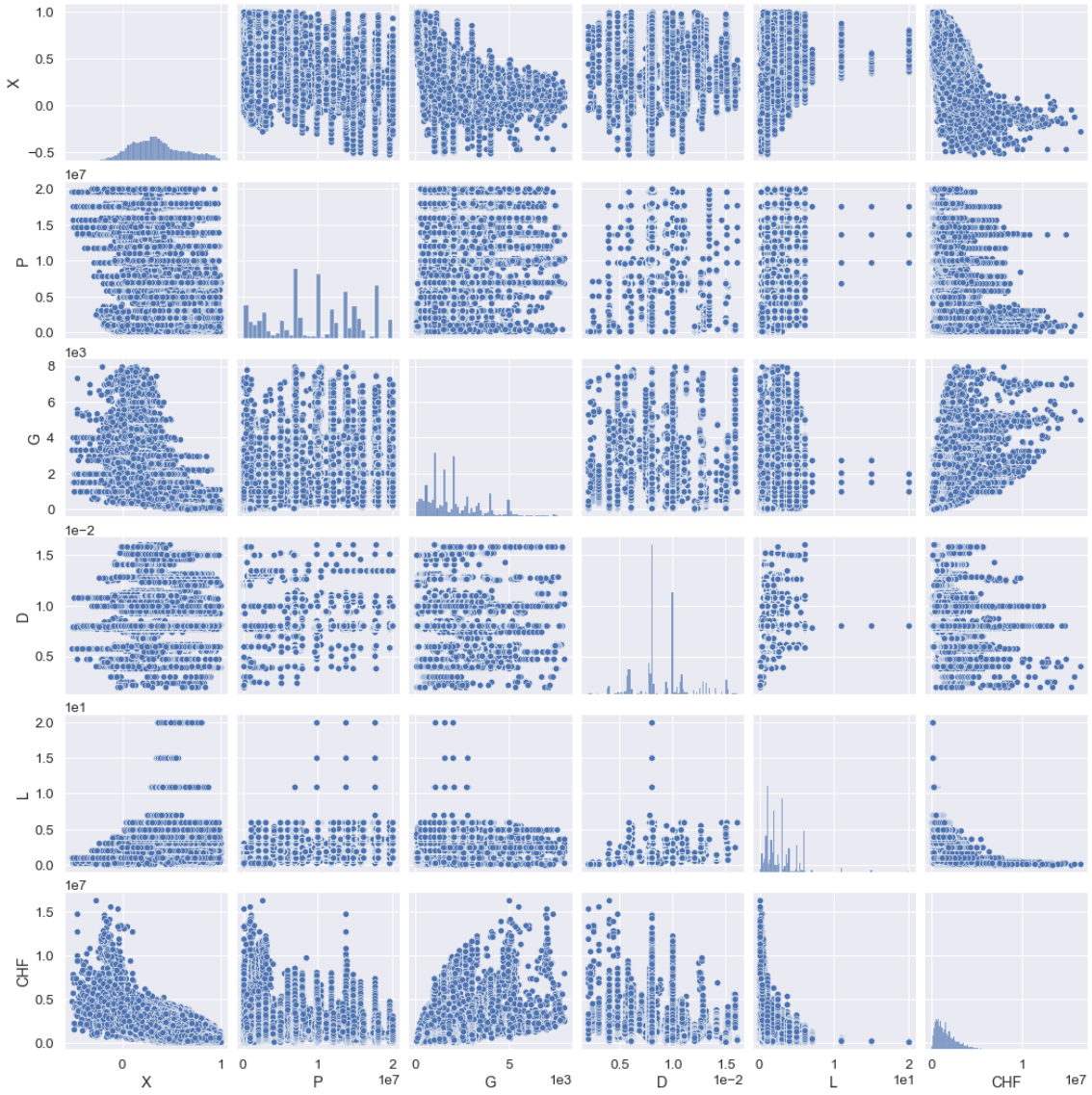


Figure 1: Distribution of data points as a function of two variables in the off-diagonal, and the proportional distribution of the points in a single variable on the diagonal, the variable is shown on the x-axis.

3.2 Data Filtering

The data file that we have described in Section 3.1 contains some irregularities, in the form of additional characters that make entries unreadable. We corrected this by hand. To make the analysis simpler, all the inputs were then converted to SI units.

To make the comparison with the LUT easier, the same data-selection criteria were used as for the creation of the 2006 LUT [3]. We remove data points that do not satisfy to following criteria:

- $100000 \leq P \leq 21000000$
- $0 \leq G \leq 8000$
- $X > 0$ and $\frac{L}{D} \leq 50$
- $0.003 \leq D \leq 0.025$
- $X < 1$
- $X < 0$ and $\frac{L}{D} \leq 25$

We then compute multiple additional properties by combining the database and fluid property data. The computed properties are: recalculated Quality at CHF occurrence (X), critical boiling number (BO_{CHF}), Weber number (WE), and the density ration (ρ_R). They can be calculated as:

$$X = \frac{H_{out} - H_f}{H_g - H_f} \qquad BO_{CHF} = \frac{CHF}{(H_g - H_f)G} \qquad WE = \frac{G^2 D}{\rho_f \gamma} \qquad \rho_r = \frac{\rho_f}{\rho_g} \tag{4}$$

Where the following are pressure-dependent values that are computed using the fluid property package *pyXSteam*: saturated liquid enthalpy (H_f), saturated steam enthalpy (H_g), surface tension (γ), saturated liquid density (ρ_f), saturated steam density (ρ_g).

H_{out} is calculated:

$$H_{out} = H_{in} + \frac{Q}{W} = H_f - DH_{in} + \frac{CHF\pi^2 LD^3}{4G} \tag{5}$$

where H_{in} is the inlet temperature, Q is the total power, and W is the mass flow rate.

Once X has been calculated we can examine the heat balance. We use the same heat balance requirement as the LUT, and remove all data points that do not fulfill the following:

$$|X_{exp} - X| \leq 0.05$$

Where X_{exp} is the quality reported in the experimental data. We will use the recalculated quality in our models to keep consistency with other calculated variables that use steam tables. When we have filtered out all data that does not fulfill the requirements above we are left with 24579 data points. This is the data that will be used in the training and testing of all our models, or a subset thereof.

3.3 Data Normalization

All the algorithms that we will use are dependent on the scales of the data. Since our different parameters have widely varying spans, as seen in Table 2, we need to normalize the input data so that our models can account for all parameters. The way we normalize the parameters can have a large impact on the end results, hence 3 normalization functions are considered to train the models.

MinMax(0,1):

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{6}$$

MinMax(-1,1):

$$x_{normalized} = 2\left(\frac{x - x_{min}}{x_{max} - x_{min}}\right) - 1 \tag{7}$$

StandardScaler:

$$x_{normalized} = \frac{x - x_{Mean}}{x_{Std}} \tag{8}$$

The MinMax functions both linearly map the variables to go from a minimum to a maximum value. These are good to use in the case where our variables are distributed in the space and do not have outliers. We test using the space $[0, 1]$ and $[-1, 1]$. The StandardScaler instead scales the data to have a mean of zero and a standard deviation of one. This is good if there are outliers since the compression of close-by points is not as pronounced as for the MinMax functions.

3.4 Transfer Data Split

When using transfer learning, we want to transfer knowledge learned from one database to another database. The purpose is to apply underlying physics learned from the base data, to a database that has similar underlying physical behavior but originates from a different, more complex system. When transferring our model it is important that the data that we are using in learning our base model, and the data we are transferring our model to, are independent. To account for this we need to split our data according to the transforms we perform.

The first split consists of removing the data from a single experiment from the base data, and then transfer to the data of that single experiment. This allows us to not change the base model significantly, since removing 100 points from our base data will make almost no difference. We can then, with the help of an arbitrary function, change the data we want to transfer to. We will use the following function:

$$CHF = CHF\left(0.9 + (\frac{G}{10000})^2\right) \tag{9}$$

This allows us to test a simplified case where the physics of the experiment has changed, for example, if the data we are transferring to has a flow mixer.

The second type of data split we will transfer to is different diameters. By training our base model on a dataset only containing diameters of $8(\pm 0.1)$ mm, and then transferring to a dataset only containing diameters of $10(\pm 0.1)$ mm, we can test how well our model can adapt to previously unseen geometries. However, this approach has the drawback that the base dataset is much smaller than our entire dataset, which can negatively impact the learning of underlying physics.

4 Algorithms

In Section 2.2 we discussed different types of algorithms used to predict CHF. Out of these, we have chosen three algorithms that we believe to be the most promising for the task. We have chosen to implement and test the following: ν - Support Vector Regression, using *Scikit-learn* [14], Gaussian process regression, also using *Scikit-learn*, and finally Fully connected neural networks, using *TensorFlow* [15].

4.1 ν - Support Vector Regression

Support Vector Machines (SVM) are a class of learning algorithms that define an optimal hyperplane in the data. A hyperplane can be seen as a high dimensional equivalent to a line in 2 dimensions, it divides our variable space into two. This means that the hyperplane will always be of one dimension less than the solution space. When using SVM for regression, the hyperplane becomes the space with which we predict. The predictions will then be the point in the hyperplane that is the closest to the data point we want to predict. How the optimal hyperplane is found depends on what type of SVM we use.

Since we are looking at a regression task, we can choose between one of the following SVM algorithms: ϵ-support vector regression (ϵ-SVR), and ν-support vector regression (ν-SVR) [16]. Both of these methods are popular prediction methods, and both ϵ-SVR [5] and ν-SVR [8] [10] have been used extensively in similar research [4]. In this previous research ν-SVR seems to perform better, we thus chose ν-SVR as our algorithm to test.

One thing all SVMs have in common is the usage of the kernel trick. As mentioned before the optimal solutions form a hyperplane, which is a high-dimensional subspace. The higher the dimension of the hyperplane, the easier it is to find good predictions. Projecting all the data points into a high-dimensional space is costly, which can be prevented by using the kernel trick. The usage of a kernel allows us to calculate distances in infinite dimensions, which allows us to calculate a hyperplane in infinite dimensions, without any projections. This means that our ν-SVR algorithm is in practice finding very high dimensional correlations.

ν-SVR utilizes so-called support vectors to build up its hyperplane. The algorithm chooses some data points and labels them as support vectors. Any points between our hyperplane and support vectors do not count towards our loss function. This means that clusters of points count less than outliers when optimizing our model. This can be good as our data is clustered in some dimensions. The hyperplane is then the plane that is perpendicular to all our support vectors.

The optimization problem for ν-SVR is defined as:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\xi}^*,\epsilon} \ & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\left(\nu\epsilon + \frac{1}{l}\sum_{i=1}^{l}(\xi_i + \xi_i^*)\right) \\ \text{subject to } & \left(\boldsymbol{w}^T\phi\left(\boldsymbol{x}_i\right) + b\right) - z_i \leq \epsilon + \xi_i \\ & z_i - \left(\boldsymbol{w}^T\phi\left(\boldsymbol{x}_i\right) + b\right) \leq \epsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0, i = 1, \ldots, l, \quad \epsilon \geq 0. \end{aligned} \tag{10}$$

Where $\boldsymbol{w}$ is the normal vector of the hyperplane, b is the offset of the hyperplane, x_i is the training vectors, ϕ maps the training vectors into a higher dimension, z_i is the target output, ϵ denotes the maximum error, l is the number of support vectors, and ξ_i is the slack variable.

C and ν are the two main hyperparameters of the algorithm. ϕ is indirectly chosen by choosing the kernel function. These three parameters can be changed manually, the rest of the parameters are defined either by the data, or while optimizing the model. The parameter ν loosely decides the number of support vectors that will be used in our solution model.

ν-SVR has the advantage of being memory efficient since we only need to save the support vectors in our model. It is also a robust model since it is defining a linear hyper-plane.

This algorithm also comes with some substantial disadvantages. Since the model is focusing on support vectors, the model is heavily dependent on the outliers to make its predictions. If the outliers in our data are only noise, and not physical outliers, it would render our predictions useless, since our model would only predict noise.

4.2 Gaussian Process Regression

Gaussian process regression (GPR) [17] is a non-parametric Bayesian type of regression algorithm. Its defining characteristic and main advantage is the fact that it uses probabilities in its estimations.

When using GPR we make the assumption that all of the probabilities can be described with Gaussian processes ($\mathcal{GP}$). $\mathcal{GP}$ are defined with a mean $m(x)$ and covariance function $k\left(x, x'\right)$. When sampling we use the prior function $f(x)$:

$$f(x) \sim \mathcal{GP}\left(m(x), k\left(x, x'\right)\right) \tag{11}$$

Here x and x' are input variables.

The training of our model is done by adding training points to our model prior f.

$$\begin{bmatrix} \mathbf{T} \\ f(x_*) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(x) \\ m(x_*) \end{bmatrix}, \begin{bmatrix} K(x,x) + \sigma_n^2 I & K\left(x, x_*\right) \\ K\left(x_*, x\right) & K\left(x_*, x_*\right) \end{bmatrix}\right) \tag{12}$$

Here x_* represents data points already accounted for, and $\sigma_n^2 I$ is both a noise term and numerical stabilizer. $\mathcal{N}$ denotes a normal distribution. $\mathbf{T}$ represent the labels to our training points, and x the input variables. We can see that the resulting covariance matrix will grow with the square of the number of data points used in our training.

When evaluating our model we will utilize the mean of the posterior, since it is the most likely outcome according to our model. When using this model for actual predictions the covariance becomes important, since that is the part that gives local probability estimates. *Scikit-learn* calculates the mean $\bar{f}_*$ and the variance $V\left[f_*\right]$ in the following way:

$$
\begin{aligned}
L &:= \text{cholesky}\left(K + \sigma_n^2 I\right) \\
\alpha &:= L^\top \backslash (L \backslash y) \\
\bar{f}_* &:= k_*^\top \alpha \\
v &:= L \backslash k_* \\
V\left[f_*\right] &:= k\left(x_*, x_*\right) - v^\top v \\
\log p(y \mid X) &:= -\tfrac{1}{2} y^\top \alpha - \textstyle\sum_i \log L_{ii} - \tfrac{n}{2} \log 2\pi
\end{aligned}
\tag{13}
$$

Here $\backslash$ is the left matrix divide. If we only look at the mean of the posterior as we do during evaluation, only the three first rows are needed. The 4th and 5th row are used to calculate the variance. The final row calculates the log marginal likelihood (LML), which is conditioned on the training data. When optimizing our model the LML is minimized. This is equivalent to choosing the parameters that make our training data the most likely according to our model. We see in equation 13 that the LML can be reduced by optimizing the α, L, and n terms. n is the number of training points. α and L depend on the K and σ_n^2 terms. K depends on which kernel function is chosen, and is then calculated by *Scikit-learn*. σ_n^2 is a hyperparameter.

We see that we need to perform a Cholesky decomposition to calculate the mean and covariance of our model. This is a computationally heavy operation that has complexity $\mathcal{O}(n^3)$.

We can thus choose three hyperparameters when building our GPR model. The number of training points n, the noise term σ_n^2, and the kernel function. We have a model that includes local probabilities, which is highly useful, but at the same time scales quite badly with the training-set size.

4.2.1 Kernels

The complexity of the GPR models can be changed by using different kernels. Using a more complex kernel will lead to a more complex model, which can lead to better predictions, but also to overfit. We will be using the radial basis function, which in *Scikit-learn* is defined as:

$$
k\left(x, x'\right) = \exp\left(-\frac{d\left(x - x'\right)^2}{2\ell^2}\right) \tag{14}
$$

Where ℓ is the length scale vector, and $d()$ is the euclidean distance vector. By changing the ℓ parameter we change how far the data points will influence our predictions. This is the most widely used kernel for GPR. *Scikit-learn* will do the hyper-parameter fitting for us, by optimizing the LML. Since the LML is not necessarily convex, the optimization will be done multiple times using different starting values. To allow the kernel length scales to fit every variable, one length scale will be fitted per input variable.

The length scale is an important parameter to optimize since it describes how far the information from a point influences the model. This defines how much each training data influences each test data.

4.2.2 K-means

As we have discussed, one of the largest drawbacks of GPR is its scaling, having a complexity of $\mathcal{O}(n^3)$. Using GPR on our whole database is thus not feasible. We need to either use a GPR approximation such as Sparse Gaussian processes [18], or reduce the number of test data. Here we have chosen to reduce the number of test data. This is done using K-means clustering.

K-Means is an algorithm that is used to define k number of clusters. The algorithm works by:

1. Randomly assigning k centers.
2. Assign each data point to its nearest cluster center.
3. Update the center to the center c of mass of all points in the cluster C: $c = \frac{1}{|C|} \sum_{x \in C} x$.
4. Repeat step 2 and 3 until the clusters no longer change.

The means of the clusters can then be used as points instead of the whole training set. By using K-means we have the advantage of reducing the number of points in the parts of our data that is denser primarily, which reduces the chance of numerical instability, while mostly keeping the sparser parts of our data. This can also be a drawback in the case where the sparse points are sparse mostly because of noise since it would increase the proportional noise in the data.

4.3 Neural Network

Neural Networks (NN) is a versatile and widely used type of machine learning. They are built up by layers of neurons, that are connected with weighted linear functions. NNs were first described as the machine equivalent of a human brain. With the rapidly increasing amount of computational power available, they have become one of the most widely used machine learning algorithms that can be used in everything from image recognition, speech synthesizing, or finding underlying functions in complex physical phenomenons.

In this thesis we have used a fully connected feed-forward NN, also called a Multi-layer perceptron. This type of NN seems complex enough to capture all the information. Applying more complex models would introduce unnecessary complexity while probably not improving the results. The fully connected feed-forward NN will be referred to as NN throughout this thesis. It is characterized by several sequential layers, each layer is built up of several neurons. Each neuron in a given layer is connected to all neurons in the previous and subsequent layer. Each one of these connections has a weight associated with it, and each neuron has a bias associated with it. The value of a neuron y can thus be described as a weighted sum of all neurons from the previous layer, and its bias b passed through an activation function f:

$$y = f(b + \sum_{i=1}^{n} w_{iy} x_i) \tag{15}$$

where x_i is the value of neuron i in the previous layer, the layer consisting of n neurons. w_{iy} is the weight between neuron i and neuron y.

4.3.1 Activation Functions

The choice of the activation function plays an important part in the performance of the NN. Using a linear activation function would make the whole network linear, and we would thus be unable to detect non-linear features. The two most used activation functions are the Sigmoid function and the Rectified linear unit function (ReLu). They are both quite simple functions that introduce non-linear behavior to our network.

$$\text{ReLu}(x) = \max(0, x) \qquad \text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{16}$$

Generally, the ReLu function is better, since it is much lighter to compute, which reduces both training and prediction time. ReLu is also generally much faster in terms of convergence [19]. It also has the advantage of not saturating at high x, since the function is not constrained in size as the Sigmoid function is, however, this characteristic can also lead to blowup, which is when the gradient becomes so large that the network becomes unstable during learning. ReLu can also have problems with dying neurons, where a large number of neurons become negative, and thus prohibit learning. The network can in that case not be optimized further.

Sigmoid is primarily used when blowup is a noticeable problem. Sigmoid has a high likelihood of leading to vanishing gradients since the gradient of the function approaches 0 when x becomes large. Vanishing gradients can be a substantial problem since it can lead to the optimization converging slowly or not at all. We will test both of these activation functions.

4.3.2 Layer Width and Depth

The Depth of the NN and the width of the individual layers are arguably the most important parameters to consider when building a NN.

The NN needs to be wide and deep enough to capture all significant underlying physics, while not overfitting the data significantly. At the same time, the model needs to be as light as possible, so that systems do not take long to calculate the model prediction, since this can be a time-sensitive problem.

Generally using many layers, also known as a deep NN is seen as more efficient and better at generalization than wider NN [20]. Having a deeper NN means that the NN can learn various levels of abstraction, which should lead to more robust models. Using excessively wide layers can also lead to memorization instead of generalization. This is when the network produces weights that can accurately predict data it has seen since it remembers it, but not on unseen data since it has not learned the underlying physics. We will thus prioritize making the model deeper, instead of wider when we try to adapt the complexity of the NN.

4.3.3 Optimization Function

Training a NN consists of tweaking the weights and biases of all neurons and connections. If the tweaking leads to better predictions we can say that the tweaking leads to a more optimized NN. The set of instructions on how to tweak the network can thus be called the optimization function.

In this thesis, we will be using the Adaptive moment estimation (Adam) optimizer [21]. This is one of the most widely used optimization functions and has multiple advantages over other optimizers: it is computationally efficient and can deal with sparse gradients and non-stationary objectives. The method is easy to implement and requires little memory. We will also try stochastic gradient descent (SGD) for comparison.

4.3.4 Loss Function

The loss function (LF) is an important parameter in defining how our network will fit our data set. The function calculates a loss, which is a distance metric between the current network output and the target value, in our case between the predicted and the measured CHF. How this distance is calculated impacts what the network prioritizes to fit.

The most widely used LF in NN for regression problems is the mean squared error (MSE):

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(Y_i - \hat{Y}_i) \tag{17}$$

This function calculates the average squared distance between the measured values Y_i and the predicted values $\hat{Y}_i$. MSE has a notable problem in our case: since it calculates an absolute error, the proportional error at low CHF will be much larger than for large CHF. We therefore also tested a LF called the Mean squared logarithmic error (MSLE):

$$MSLE = \frac{1}{n}\sum_{i=0}^{n}(log(Y_i + 1) - log(\hat{Y}_i + 1)) \tag{18}$$

Using the logarithm of the measurements and predictions allows us to measure their relative distance, which in theory should lead to a higher absolute error, but lower proportional error for our model.

4.3.5 Regularization

A well-known problem with NN is overfitting. To decrease overfitting the usage of some type of regularization has become standard. We will test two types of regularization: L2-regularization and Dropout. L2-regularization consists in adding a term to the loss function that penalizes large values during training:

$$loss = \mathrm{L}_2 \sum_i x_i^2 \tag{19}$$

Where L_2 is a parameter chosen by the user. We choose what x_i represents, depending on what we want to regularize, it can be either the values of the kernels, the biases, or the activity of the layer. By penalizing large values in these variables the risk of overfitting decreases. However, if we choose a large L_2, the risk of underfitting increases.

Dropout works by setting some incoming weights to zero during one training step. How many weights are set to zero during training is decided by a *rate* parameter. For example, if the rate is set to 0.1, the nodes will be set to zero with a frequency of 0.1. The other nodes have their output increased to keep the sum of all outputs unchanged.

Both of these methods are applied to single layers and can have different parameters in different layers.

4.3.6 Training Time

One very limiting factor on how much hyper-parameter optimization can be performed on a NN is how long it takes to train one set of parameters. One obvious way to decrease the training time of a NN is to reduce its size, and thus the number of parameters to optimize. This is however not always feasible, as one might need a complex network to accurately model the underlying function. We must then use other ways to reduce training time, such as reducing how much the network has to train. This can be done in two complementary ways: utilizing Early stopping and Learning rate decay.

Early stopping consists in defining a condition that, if met, interrupts the training of the network. This ensures that we do not train the network unnecessarily if it has already converged, which means that the model overfits less. This is because it does not have more time to optimize the training set when the optimal model for the validation set is achieved. It also allows us to set a large maximum training time, which means that we have less chance of the model not reaching its optimal state.

Learning rate decay is a method in which a larger training rate can be defined at the beginning since the model will then be far away from an optimal state, and decrease the learning rate when we get closer to an optimal solution. This leads to smoother learning and reduces the chance of overshooting during the optimization while speeding up the optimization by approaching the optimal state faster in the beginning. Here we will be using exponential decay since the training can take a large number of epochs and may vary between models.

4.3.7 Transfer Learning

When training a NN on a data set, we hope to capture the underlying physical functions that lead to the data in the data set. In our case, the data only describes the occurrence of CHF in round tubes, using uniform heating, and water as coolant. This does not represent how an actual nuclear reactor works, but should, in theory, be subject to similar physical behavior. Changing for example the coolant should change some variables in the underlying functions, however, the structure of the physical function should stay the same. Using transfer learning, we should then be able to transfer the physical functions that our NN has learned, and adapt them to our new variables.

To do this, we need to train a NN on our large database, and then transfer this to a smaller database from a different, more complex system. In practice, this is done by freezing the hidden layers of the trained NN so that they do not change during training, removing the output layer, and then adding one or more hidden layers at the end of the NN and a new output layer. The final layer/layers are then trained to adapt to the more complex system.

If necessary, the final NN can also be fine-tuned, by unfreezing some or all layers, and then training the model with a very small learning rate. This can for example adapt the base model slightly to the new physics, for example in the case where the base model slightly overfitted the base data.

If the NN has captured the correct underlying physics, and the database we want to transfer to is large enough, we should be able to get a well-functioning model for the physics in the problem we want to transfer to.

4.4 Hyperparameter Optimization

For all the algorithms that have been described in Section 4, multiple hyperparameters need to be tuned for optimal performance. These were all tuned using *Optuna* [22], a framework-agnostic hyper-parameter optimizer. *Optuna* is a good choice in our case since it can be used for all frameworks, that is both for *Scikit-learn* and *Tensorflow*. It also has the advantage of being easy to parallelize, which speeds up training, and has automatic hyper-parameter search, within a given parameter space.

To find the optimal hyper-parameters *Optuna* uses the Tree-structured parzen estimator algorithm [23]. This is a Bayesian model-based optimization method, which builds a probability model of our objective function, it then proposes a set of hyper-parameters to evaluate next. The algorithm decreases the number of configurations to test, compared to random search, and increases the achieved best results [24].

The probability distribution of the model is divided into a "good" distribution $l(x)$ and a "bad" distribution $g(x)$. These distributions are built using the knowledge we have already gathered, by evaluating the objective function at different hyper-parameter sets. The new hyper-parameter sets are then chosen to maximise $\frac{l(x)}{g(x)}$. The modeled distributions are updated after each evaluation, and should in theory improve the next guess. Calculating the expected improvement, the algorithms can also balance exploration versus exploitation, that is when to improve on existing predictions, by using our knowledge of the distributions, and when to guess where other better predictions could be.

Table 3: When optimizing each algorithm we minimize a measurement, by optimizing the parameters using *Optuna*

Algorithm	Measurement	Parameters
SVR	MSE	C, ν
GPR	MSE	σ_n^2, k
NN	MSLE & Size	w, hl

In Table 3 we can see the measurement the objective function minimizes, and the hyperparameters that *Optuna* is tuning: C, ν are the hyperparameters defined in Section 4.1. σ_n^2 and k are defined in Section 4.2, where k is the number of clusters, and thus the number of training data points. w is an array of the width for each NN layer, and hl is the number of hidden layers in the NN.

We will tune our hyper-parameters by using a fixed number of hyper-parameter trials. If the models improve in the latter part of the trials they are continued. If they do not improve in the last 50 trials the best hyper-parameter set is assumed to be very close to optimal. All the sets were trained for approximately 100-250 trials. The optimizations are done using 5-fold cross-validation (CV) to improve the generalization of the hyper-parameters.

5 Implementation

5.1 Software

All of the implementations are done using Python 3.10. The implementations of GPR and ν-SVR were made with *Scikit-learn*, and the implementation of the NN was made using *Tensorflow*. The optimization of the hyper-parameters for all models was made with *Optuna* and sped up by parallelizing the optimization using *Joblib*. The visualization of the models and the data was done using *Matplotlib* and *Seaborn*. The interpolation of the LUT for comparison was done using *Scipy*. The fluid properties were gathered from *pyXSteam*. The *Pandas*, *Numpy*, *Math*, and *Random* packages were also used during computations.

5.2 Computations

First, all the data was treated as described in Section 3.2. Then we looked at optimizing the hyper-parameters for each algorithm. To do this, each algorithm was implemented as an objective function in *Optuna*, meaning that the function had the hyper-parameters defined by *Optuna* as input parameters, and a score as an output parameter. The score for the GPR and ν-SVR was the average MSE. For the NN both the MSLE and the number of network parameters, so-called size, was used as scores. The models were tested using CV, and the score was the average of the scores from each CV fold. This was so that the hyper-parameters would be more generalized than if we look at a specific part of the data. The CV was randomly initialized.

Each method was then implemented separately using the optimized hyperparameters. The implementations were all done with the same data split for the training and test sets, to allow for performance comparisons. Each model was then plotted and the accuracy of the models was compared to investigate which of the models was best suited for the predictions.

It was quite clear that NN performed the best, and was thus chosen as the algorithm to continue with. More extensive visualization efforts were made to guard against overfitting and to allow for the analysis of the inner workings of the model.

Finally, some transfer tests were performed using the NN to test the feasibility of transfer learning. This was done by using the data splitting discussed in Section 3.4. First, a base model was trained on the base data. This model used the same hyper-parameters as the optimal model for the complete dataset. Then the model was transferred to the transfer data, as described in Section 4.3.7.

6 Results

There are numerous ways to measure the performance of regression algorithms. Here we will present the mean and standard deviation of the M/P factors (Std). We will also present the normalized root mean square error (NRMSE), which is calculated in the following way:

$$NRMSE = \frac{\sqrt{\sum_{i=0}^{N}(Y_i - \hat{Y}_i)^2/N}}{Y_{mean}} \tag{20}$$

The mean of M/P measures whether the model is biased. If the mean deviates from one significantly, the model has probably not found the right fit. The Std measures the proportional error of the model since each model investigation is done as a factor. This is the most important parameter to consider since it will also be how the model performance will be evaluated if we want to put the model into production. Finally, the NRMSE is a measurement of the absolute error. This means that the errors at high CHF will count more towards the NRMSE than the same proportional error at lower CHF. NRMSE is an often-used metric in the field, and can thus also be used for comparison with previous research.

Using T_{in}, P, G, D, L as inputs often give better results, they are included for comparisons sake. However, these models are very hard to utilize in practice, since T_{in} does not have a direct translation in more complex geometries. Additionally, if non-uniform heating is used, this must be added to the model.

We therefore use X, P, G, D, L as our benchmark input parameters. This also has some problems with the geometric parameters, especially the L parameter. Therefore we will also test models using different input parameter combinations when using the NN: X, P, G, $\frac{L}{D}$, as well as the combination: X, P, G, D, that is, a model based on local parameters only.

The data discussed in Section 3.1 is divided into two datasets. The training dataset contains 80% of the points, while the test dataset consists of the remaining 20% data, roughly 5000 points. This is so that the data we use to test the performance of the models have not been used to train the model. The number of points in the test set is important to remember when looking at Figure 2, 3, 4, since the outliers are more visible than the large amount of data within the green lines representing a 10% error.

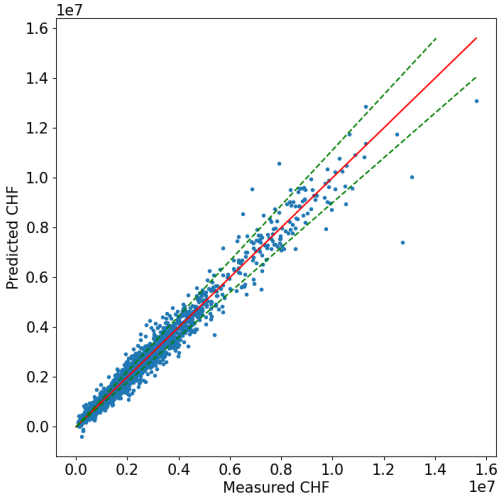
6.1 ν - Support Vector Regression

When optimizing the ν-SVR algorithm we looked at the hyper-parameters C, ν, and the normalization of the data. We optimized one set of hyper-parameters for each normalization type. The performances of these models are shown in Table 4. We see that as mentioned before, the parameter using T_{in} as input has a lower NRMSE, however, using X instead is more interesting in a practical sense. Additionally, the most interesting performance indicator is the Std, a metric in which the model utilizing X performs better. We see in Table 4 that the model using the Standardscaler is performing the best in all our measurements. We conclude that the most interesting ν-SVR model is the one using the input parameters: G, P, D, L, X, the Standardscaler, $C = 7.96$, and $\nu = 0.47$.

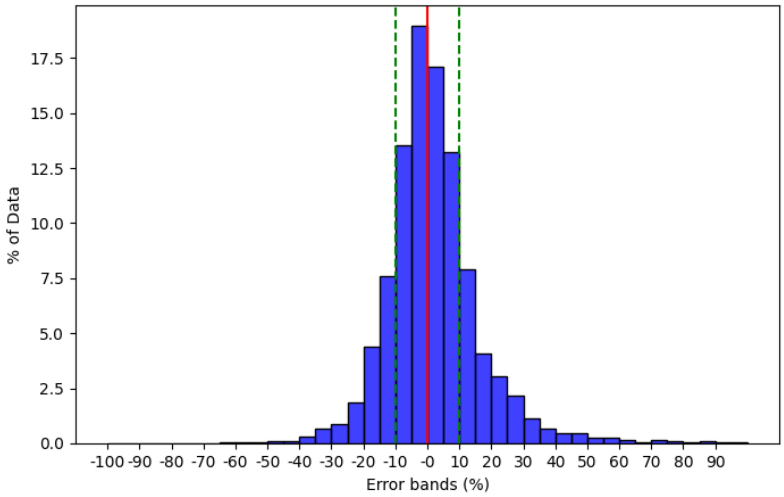
Table 4: Results for ν-SVR, using different variable and scaling options, and the coupled optimized C and ν hyper-parameters

Variables	Scaling	C	ν	NRMSE	Std	Mean
G, P, D, L, X	MinMax(-1,1)	1.45	0.37	0.157	0.218	1.028
G, P, D, L, X	MinMax(0,1)	1.43	0.46	0.152	0.21	1.021
G, P, D, L, X	Standardscaler	7.96	0.47	0.142	0.177	1.021
G, P, D, L, T_{in}	Standardscaler	7.85	0.28	0.11	0.21	1.013

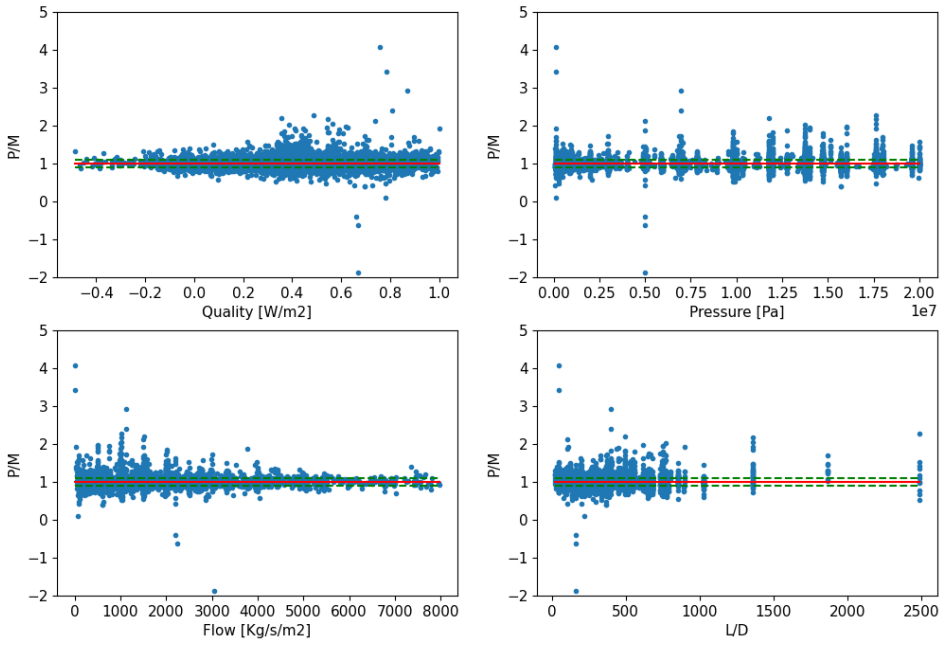
We take a closer look at the performance of the optimal ν-SVR model in Figure 2. In 2a, we see that there are some negative CHF predictions at low measured CHF, which is nonphysical. In 2b, we notice a larger than expected bias towards positive error. Finally, in 2c we see that there are parameter-dependent errors in all parameters, except the pressure.



(a)



(b)



(c)

Figure 2: Predictions of the optimal ν-SVR model. The red lines represent perfect predictions, and the green lines represent 10% error.

6.2 Gaussian Process Regression

The GPR models are chosen and optimized in a similar way to the ν-SVR. The set of hyper-parameters σ_n^2 and k are optimized individually for each scaler. The length scales are an output of the algorithm.

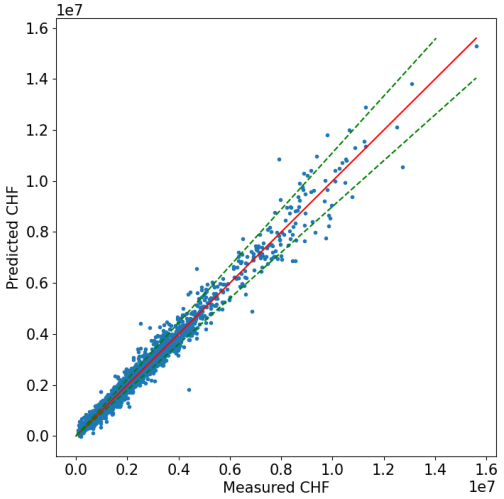
We see in Table 5 that the length scales are inconsistent between models. Changes in the length scales are expected for different data scaling and are quite consistent between runs for the same scaling. Between MinMax(-1,1) and MinMax(0,1), we expect consistent changes for every variable since we simply double the length between each point, however, we do not notice this. This might explain why the MinMax(0,1) scaling is performing so poorly.

We also see that the T_{in} model is performing well, however, we are once again discarding it for practical reasons. We see that as for the ν-SVR, the model using the Standardscaler is performing the best in terms of Std. We conclude that the most interesting GPR model is the one using the input parameters: G, P, D, L, X, the Standardscaler, $\sigma_n^2 = 6.8\text{e}{-3}$ $k = 2615$, and resulting in the length scale: [1.12, 0.477, 0.785, 2.42, 0.782]

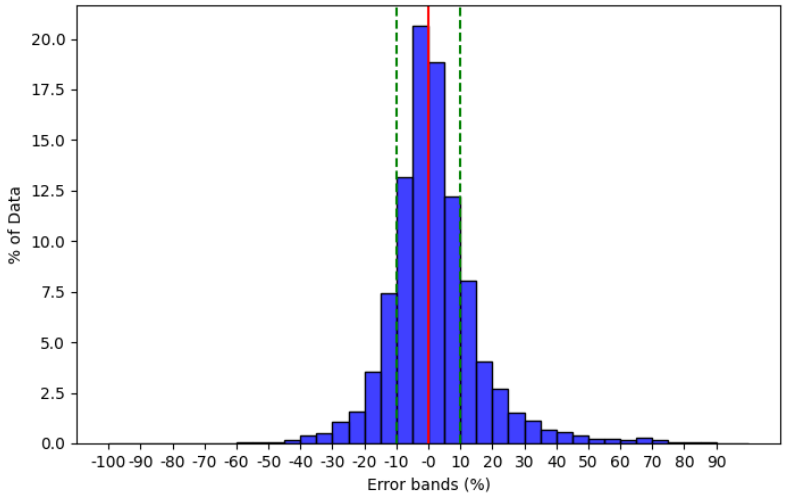
Table 5: Results for GPR, using different variable and scaling options, the final length scales, and the coupled optimized σ_n^2 and k hyper-parameters

Variables	Scaling	Length scales	σ_n^2	k	NRMSE	Std	Mean
G, P, D, L, X	MinMax(-1,1)	[0.737, 0.479, 0.298, 1.23, 0.485]	3.83E-04	2658	0.127	0.188	1.025
G, P, D, L, X	MinMax(0,1)	[0.516, 0.44, 1.38, 0.622, 0.264]	4.54E-04	2797	0.149	0.241	1.028
G, P, D, L, X	Standardscaler	[1.12, 0.477, 0.785, 2.42, 0.782]	6.80E-03	2615	0.127	0.177	1.022
G, P, D, L, T_{in}	Standardscaler	[1.32, 0.878, 2.87, 0.275, 1.51]	2.85E-03	2577	0.093	0.148	1.01

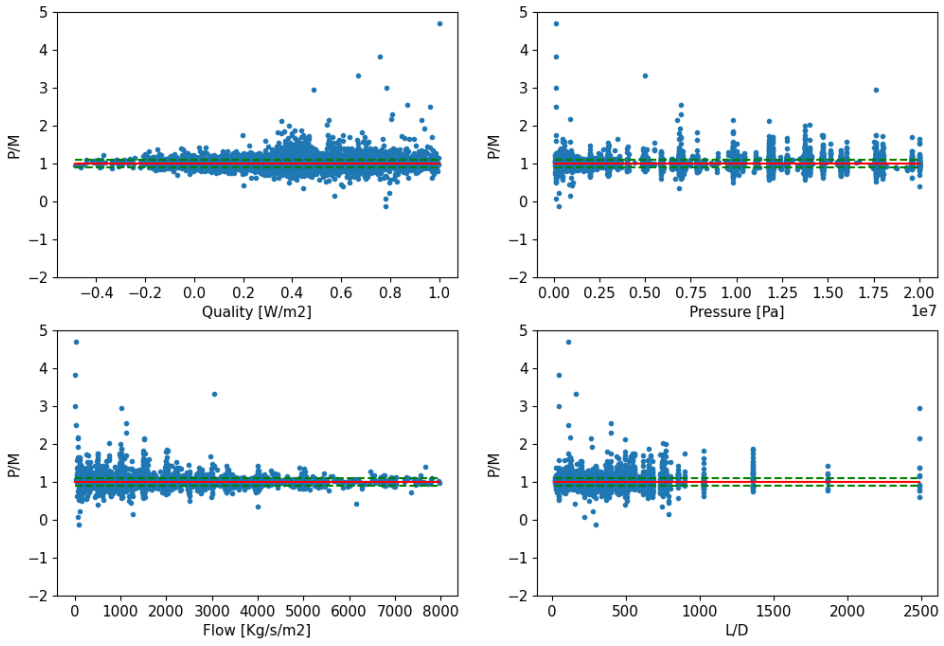
When looking at the performance of the optimal GPR in Figure 3, we see similar results to the optimal ν-SVR model. If 3a we see negative CHF predictions, however only very few. In 3b we see a similar non-centered error distribution, however, this also seems to be an improvement on the ν-SVR model. The problem of the variable dependent error as a function of parameters however does not seem to have improved, as seen in 3c.



(a)



(b)



(c)

Figure 3: Predictions of the optimal GPR model. The red lines represent perfect predictions, and the green lines represent 10% error.

6.3 Neural network

The NN is much more complex to optimize than the previous two methods. There are many more hyper-parameters to test, and we cannot only look at the error since that could lead to huge networks. We therefore first tested simpler parameters by comparing the results manually when changing parameters with fewer options, such as the loss function, optimization function, and activation functions. The tweaking of the training time and the regularization was also done by hand. Finally, the NN depth and width were optimized using Optuna.

The Sigmoid function was unable to converge, and the results were thus useless. Due to this, and its other advantages discussed in Section 4.3.1, we went with the ReLu activation function. For the regularization, we decided to go with dropout on the two first layers, since this did not decrease performance, which using $l2$ regularization or more dropout led to. Using dropout still decreased the overfit substantially. The training rate was set to 0.001 and decreased exponentially with the factor of 0.96 approximately every n*2 batches, where n is the number of data points in the training set. This is approximately every 32 epochs. The training was set to end when the validation loss did not decrease for 50 epochs in a row. Then the model with the lowest validation loss was selected as the trained model.

We see in Table 6 that the MSLE loss function decreases the Std substantially and that it is more suitable for our usage. Using the Adam optimization function leads to faster learning, at the same time SGD never reached the same minimum as Adam. When it comes to variables, the usual variable combination G, P, D, L, X is performing the best, however, we see that the combination G, P, $\frac{L}{D}$, X is also performing well in the case where we want to reduce the number of parameters, or if we want to have a non-dimensional geometric parameter.

Table 6: Results for NN, comparing different variables, layer architecture, optimization functions, loss functions, and the number of parameter in the network, denoted as size.

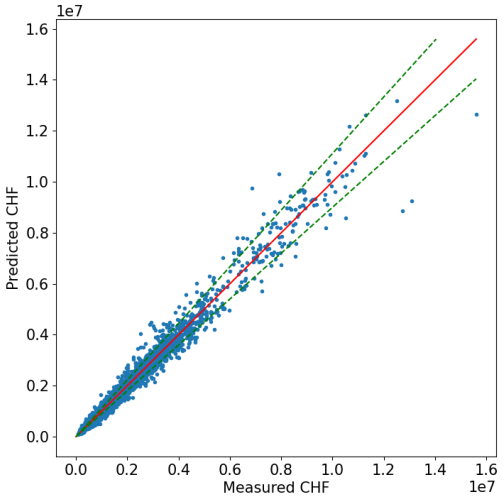
Variables	Layers	Opt. fun.	Loss fun.	Size	NRMSE	Std	Mean
G, P, D, L, X	[5,63,30,30,22,37,18,30,1]	Adam	MSLE	6057	0.148	0.128	1.011
G, P, D, L, X	[5,63,30,30,22,37,18,30,1]	Adam	MSE	6057	0.122	0.174	1.033
G, P, D, L, X	[5,63,30,30,22,37,18,30,1]	SGD	MSLE	6057	0.393	0.460	1.050
G, P, D, X	[5,63,30,30,22,37,18,30,1]	Adam	MSLE	6057	0.170	0.243	1.020
G, P, $\frac{L}{D}$, X	[5,63,30,30,22,37,18,30,1]	Adam	MSLE	6057	0.187	0.159	1.015
G, P, D, L, X	[5, 39,40,31,42,1]	Adam	MSLE	4503	0.150	0.135	1.005
G, P, D, L, X	[5,61,51,28,39,26,21,20,14,1]	Adam	MSLE	8482	0.134	0.123	1.013

When looking at the layer architecture, we need to both optimize the predictions of the model, while keeping the model reasonable in size. To achieve this, we set a dual goal for *Optuna*, to minimize the loss, while also minimizing the size, or the number of trainable parameters, of the network. This leads us to multiple well-performing models. We can choose if we prefer performance or size, and select one of them consequently. We decided to choose the best-performing model as the optimal model, but we also believe the model of size 6057 is performing well enough to be chosen if the size of the network is a more important parameter.

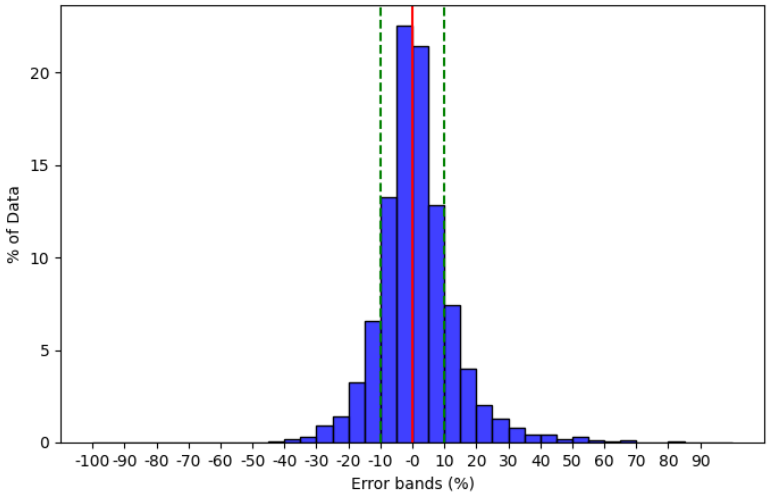
Multiple other input variable combinations were tested. Using the variables discussed in Section 2.1, such as We or ρ_r, lead to poor performance. One reason why this might be is the distribution of the parameters, which are not optimized for non-dimensional parameters. The input combination G, P, $\frac{L}{D}$, X was the only one with computed variables that could be compared to the models with the standard variables. Using the BO_{CHF} as output resulted in non-converging models.

The optimal model we have is then defined by: the input variables G, P, D, L, X; the layer architecture [5,61,51,28,39,26,21,20,14,1], where the first layer is performing normalization, the two subsequent layers have dropout of 0.01 performed on them, and the final layer is the output layer; the Adam optimization function; the MSLE loss function; and the ReLu activation function.

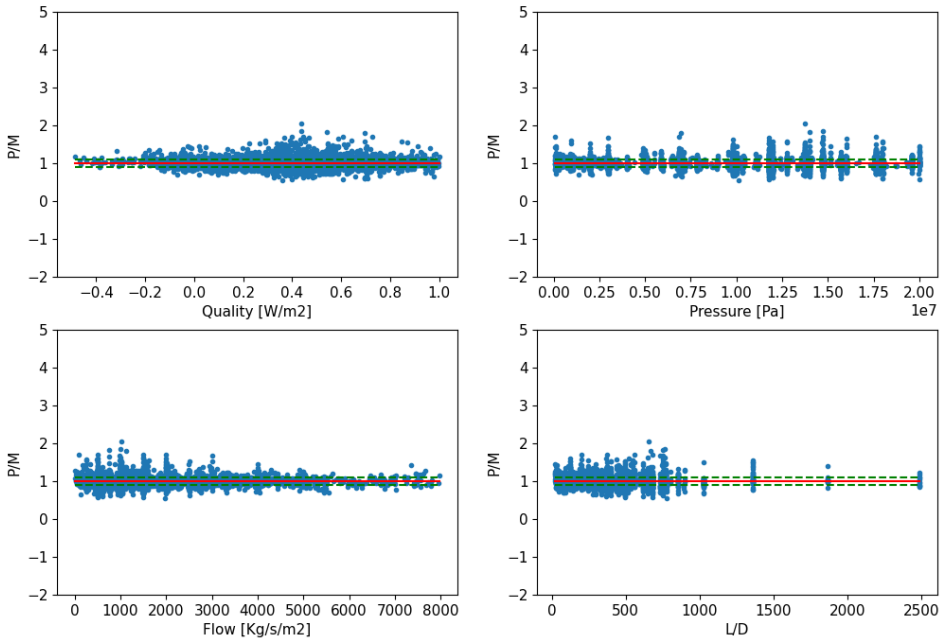
We look at the predictions of the optimal NN model in Figure 4. In 4a, we see that this model is performing much better than the two previous models at low measured CHF. The error distributions are again non-symmetric, as seen in 4b, however, they are an improvement on the



(a)



(b)



(c)

Figure 4: Predictions of the optimal NN model. The red lines represent perfect predictions, and the green lines represent 10% error.

two previous models. Finally, we can see in Figure 4c that the variable-dependent biases have decreased, and are less clear for the pressure and geometric parameters.

6.4 Algorithm Comparison

Now that we have optimized models for all algorithms, we can compare them on an equal footing. We look at their NRMSE, Std, and Mean for both the test-set, train-set, and the full data-set. We see in Table 7, that our models perform better than the LUT in all our metrics. The GPR model has the best NRMSE, however, it is only marginally better than the NN model. The NN model has both the best Std and mean. The Std is substantially better than the other models. The LUT performs better on the test data than on the train data, while our models do the opposite. Since this difference is the same for all models we assume that this is acceptable, this is further investigated for the NN in Section 6.5. All our models have comparable differences between data sets.

The better performance of the NN model in the Std measurement is partly due to the different loss functions. The MSLE loss function can also be utilized for the ν-SVR and GPR models, however, it is more complicated to implement, and was not attempted since the NN outperforms the other models when utilizing the MSE as well.

Table 7: Method comparisons. The data from the testset contains 20% of all data, and the data from the trainset contains the remaining 80%.

	NRMSE			Std			Mean		
Method	test	train	all	test	train	all	test	train	all
LUT	0.211	0.217	0.216	0.355	0.383	0.377	1.035	1.035	1.035
ν-SVR	0.142	0.137	0.138	0.177	0.167	0.169	1.021	1.018	1.018
GPR	0.127	0.117	0.119	0.177	0.162	0.165	1.022	1.020	1.020
NN	0.134	0.124	0.126	0.123	0.113	0.115	1.013	1.009	1.010

6.5 Model Behaviour

It is quite clear that the NN model is performing the best out of the models presented in Table 7, and we will thus look deeper into the optimal NN model. To get a visual representation of the NN we will discuss in this section we can look at Figure 5. We see a NN consisting of 8 hidden layers of mostly decreasing width, with 5 input variables and a single output.

First, we compare the errors between our NN and the LUT. We see in Figure 6 that the error has decreased significantly, both overestimations and underestimations have been reduced. We also notice that the error has decreased more in the underestimations (negative error bands). The underestimations that are off by more than 40% have disappeared. For the overestimations, they are reduced to under 70%. In terms of absolute error, this is equivalent.

To look further at whether the model is overfitting, we can look at the learning loss. We see in Figure 7 that the train and validation loss are similar during the whole training. The test done at the end also has similar errors to the two other sets at the end. This is an indication that the dropout used is sufficient to prevent serious overfitting.

Let us now look more closely at the predictions the model makes. We will look at the prediction structure one parameter at a time. All the points shown represent similar data points from the test dataset.

We want to compare our results to the LUT, and will thus use one parameter as a sliding parameter, to represent the added input dimension. This sliding parameter will be the length, since it is the added parameter compared to the LUT. This is the case in all plots except the one where length is the main parameter (Figure 11), in such case the quality will be used as the sliding parameter. The color of the data point corresponds to the model prediction for a specific sliding parameter. The data is the within 5% of the sliding parameter of the model predictions. The different sliding parameters can be seen in the legend of each plot.

The parameters that are neither the main parameter of the plot nor the sliding parameter, are frozen parameters. All the predictions will be done using these frozen parameters, and the data

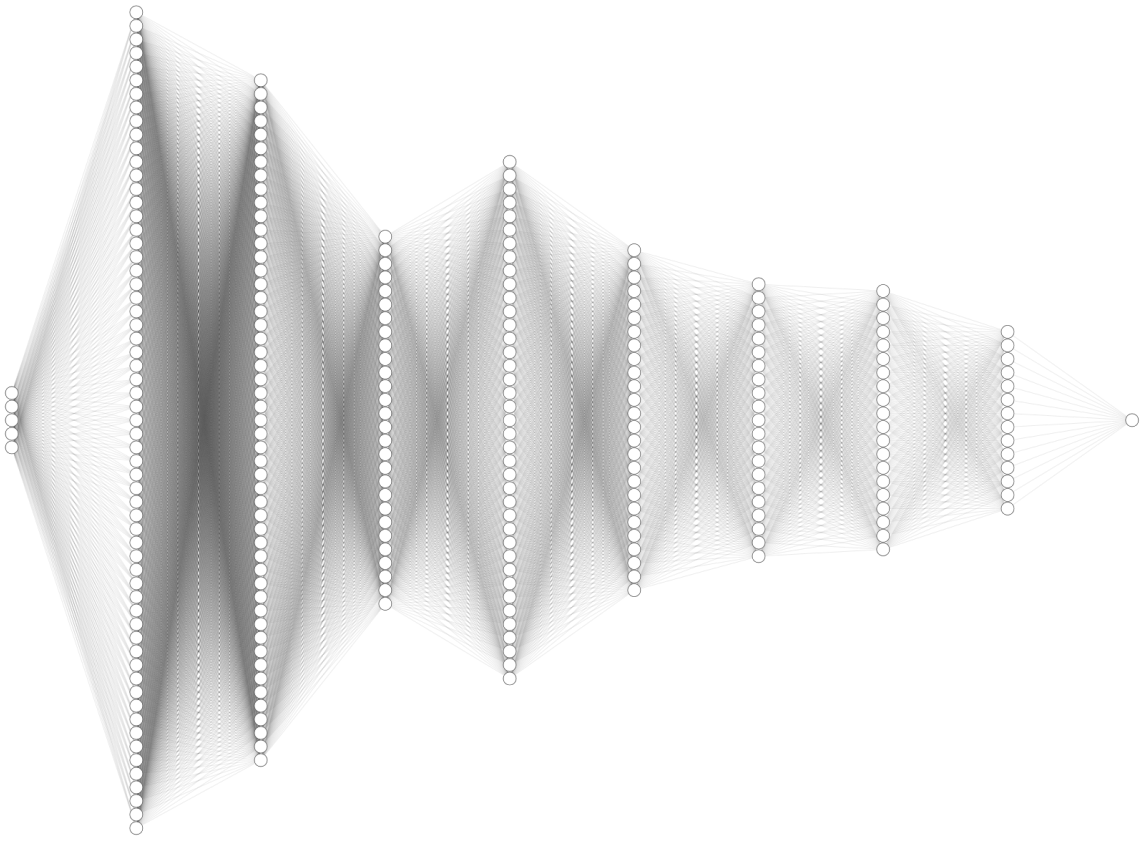


Figure 5: Visual representation of the node distributions in our optimal NN

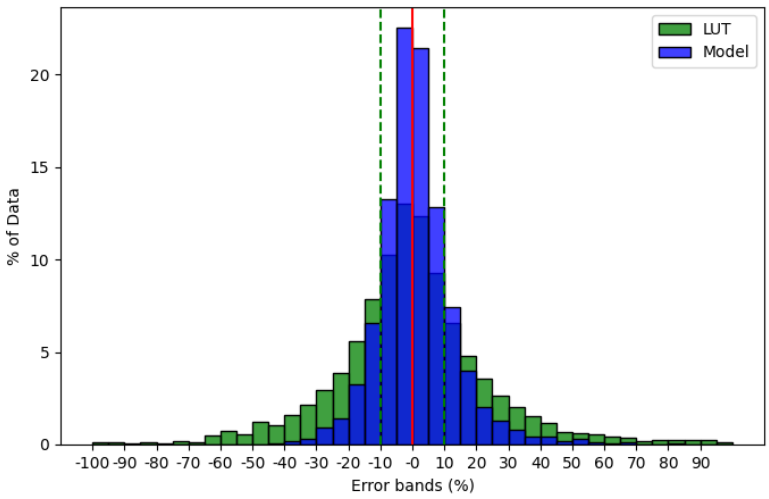


Figure 6: Comparison Model and LUT errors

points represented are all within 10 % of the frozen parameters. In the case where quality is a frozen parameter, the margin is instead always of 0.1, as the parameter can be close to zero which would result in small or undefined margins. The frozen parameters can be seen in the title of each plot.

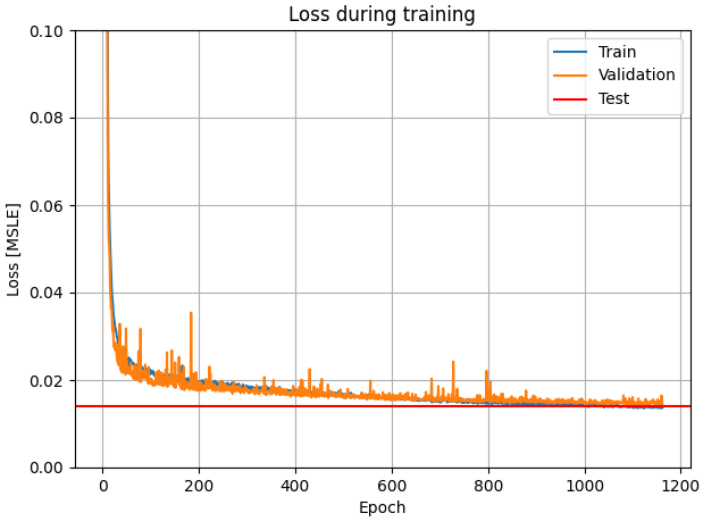


Figure 7: Loss during training

We can thus visualize the model prediction changes for each parameter individually, visualize the improvements made over the LUT, especially the influence of an additional parameter in the form of sliding parameters, and how well they predict similar data points.

Figure 8 shows CHF vs Mass Flux for our model, the LUT, and the corresponding points for a selected example where P = 9.8 MPa, D = 8 mm, X = 0.29. We see that the data has a large CHF variance for similar mass flux. When the data is partitioned in different lengths the variance decreases. This implies that the data variance is dependent on the variance in the sliding parameter. We see that our model predictions are much better at predicting the data points that are of similar length, compared to the LUT. Our model is following the same structure as the LUT, with a sharp increase of CHF at low mass flux, followed by a reduction, and finally a steady increase of the CHF. However, this structure has different amplitudes and offsets for each length.

Figure 9 shows CHF vs Pressure for our model, the LUT, and the corresponding points for a selected example where G = 1510 kg/m2s, D = 8 mm, X = 0.40. We see similar performance as for the mass flux, with high variance in the data, but lower variance within a defined length. Our model is performing much better when the length parameter is accounted for and manages to predict the data points with the same length well. Here too our model follows a similar structure to the LUT.

Figure 10 shows CHF vs Diameter for our model, the LUT, and the corresponding points for a selected example where G = 1510 kg/m2s, P = 11.8 MPa, X = 0.40. We see very large variances in CHF for the same diameter. Our model, even with the added length parameter is not able to predict the datapoint with very high CHF. This might be an indication that the diameter is a less influential parameter, and that the variance of the other parameters the data points are selected from has a larger influence on the results than the change in diameter. We see that the structure is again generally similar to the LUT.

Figure 11 shows CHF vs Length for our model, the LUT, and the corresponding points for a selected example where G = 2030 kg/m2s, P = 13.5 MPa, D = 8 mm. The length parameter, which is not accounted for in the LUT, does not seem to be a secondary parameter, as stated by Groenveld [3]. We see that the length parameters have a clear effect on the data. Our model can predict the data well when accounting for the extra quality parameter. The CHF is decreasing faster at low lengths and becomes more constant at lengths of over 10 meters.

Finally, let us look at the influence of the quality. In Figure 12 we see that for high qualities the data has a large variance. Our model is better at predicting the data when accounting for the length parameter, however, the improvement is less clear than for the length, pressure, and mass

flux. Our model once again has the same general structure as the LUT. We also see questionable predictions at X = 1, where CHF by definition should be zero. We see that this is not the case. This might be due to how quality is measured: it is the point where all liquid has evaporated at equilibrium. If the CHF occurs before the system has reached equilibrium some liquids may remain, which would prevent CHF = 0. For low qualities, as seen in Figure 13, the data has much lower variance. Here the model does not show any clear improvements over the LUT. The model is also performing in unexpected (i.e. non linear) ways at negative qualities. This is probably due to the sparse data at very low qualities, where the model is hence expected to have higher uncertainties.

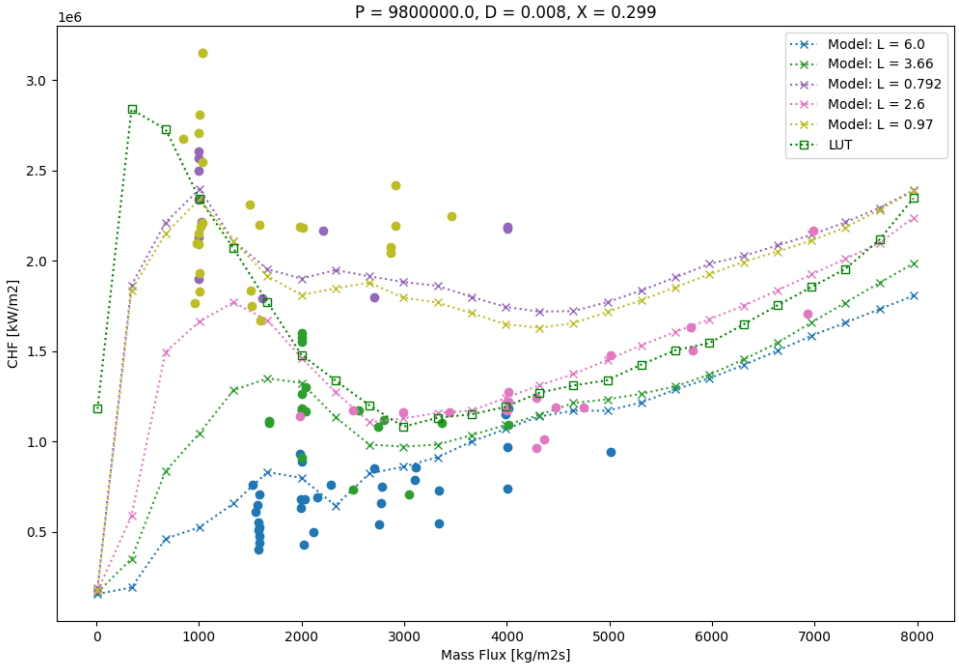


Figure 8: The behavior of the model and LUT as a function of mass flux.

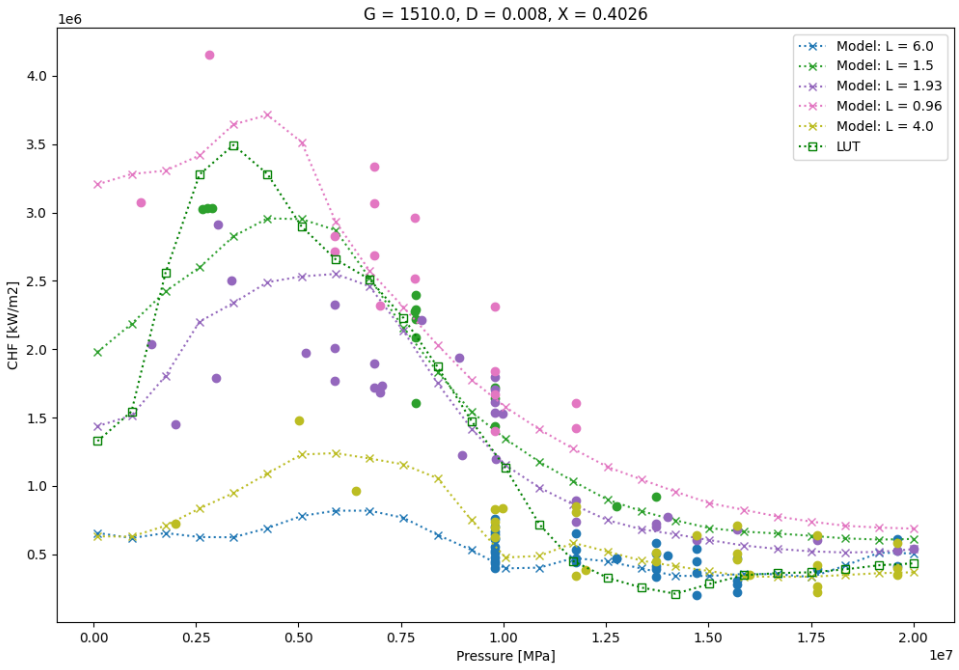


Figure 9: The behavior of the model and LUT as a function of pressure.

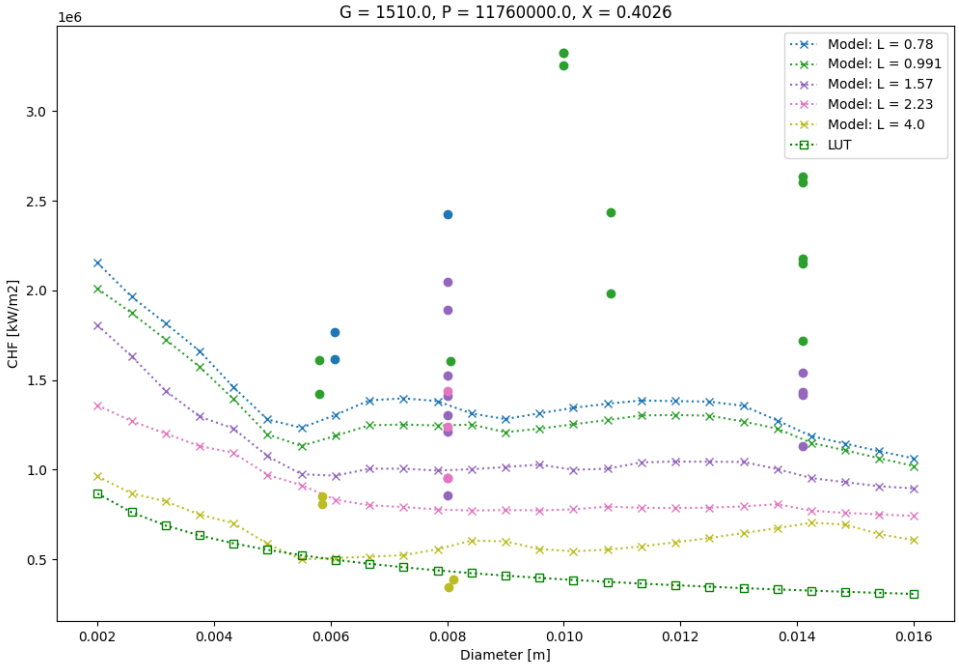


Figure 10: The behavior of the model and LUT as a function of diameter.

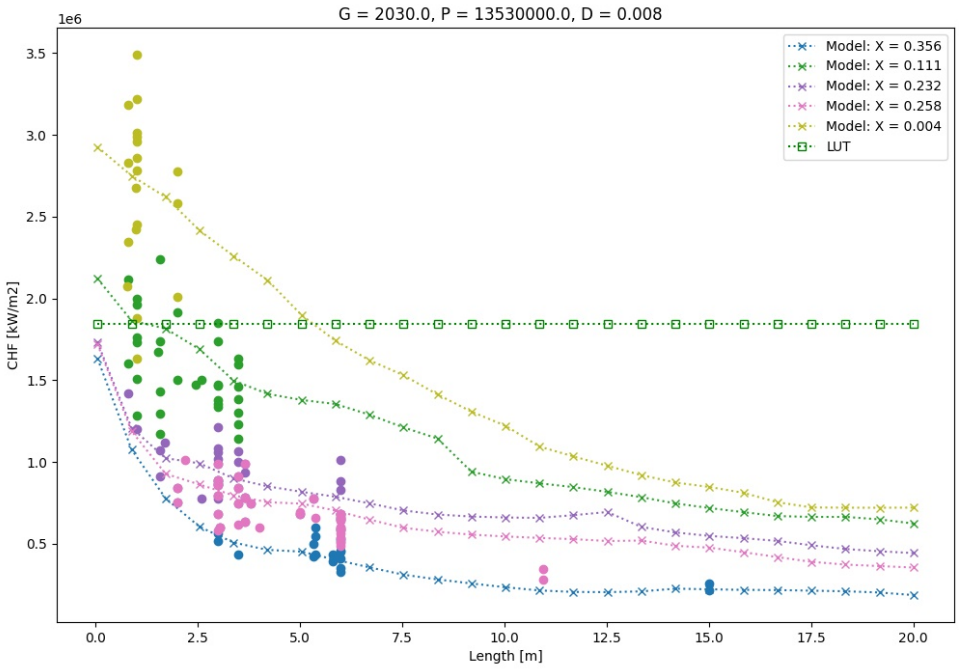


Figure 11: The behavior of the model and LUT as a function of length.

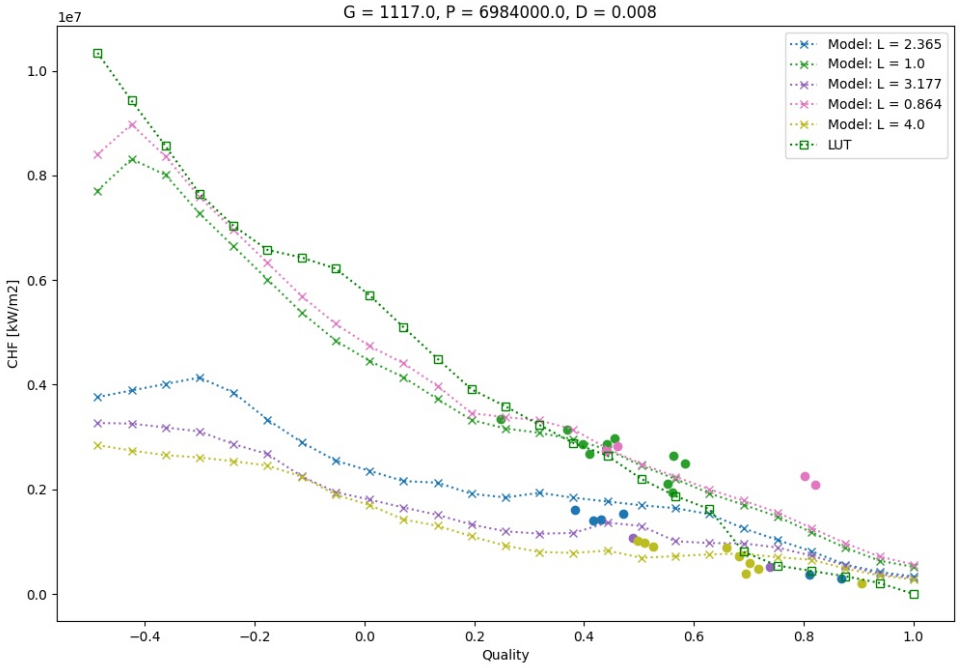


Figure 12: The behavior of the model and LUT as a function of quality, for high qualities.

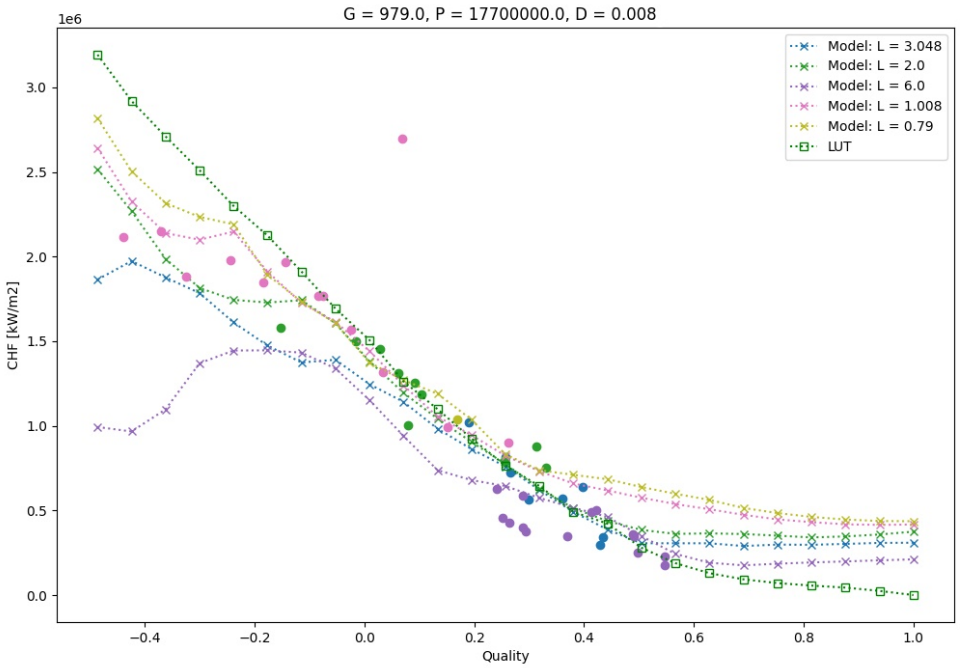


Figure 13: The behavior of the model and LUT as a function of quality, for low qualities.

6.6 Transfer Learning

Since the NN was the best performing algorithm, we will here again only work with NN. When using transfer learning we need a base model, this will be the optimal NN hyper-parameters from Table 6, however, this time it will be trained on only a part of the data. We will then transfer it to some other unseen data. The Data in this section is described in Section 3.4.

6.6.1 Transfer to Altered CHF

First, we look at the transfer from a base model identical to the optimal NN model described in Section 6.3, with the only difference that one of the databases was removed from both training and testing. The removed database is entitled "Groeneveld 1985", containing 116 data points. The data was altered using Function 9. The transferred model was then trained by freezing the base model and replacing the output layer with two layers of 40 neurons each and a new output layer. We use the same optimal function combinations discussed in Section 6.3. The training is done on 35 data points and the testing on 81 data points.

We see in Table 8 that the base model has nearly the same performance as the optimal NN from Section 6.3 when using the base data, which means that the NN is not changing significantly compared to when all the data is used. We see that the mean of the transferred data on the base model is far from one, which means that the transferred data has been changed sufficiently, which justifies using transfer learning. We also see that the Std is lower than for the base model, which is explained by the fact that the database used for transfer has a low Std overall. After performing transfer learning the mean has been corrected, and the Std has been reduced. We can see in Figure 14b that the error is not symmetric. When looking at Figure 14a we see that the large errors come from large and small CHF. In Figure 14c we see that there is a small mass flux bias, with under predictions for high mass flux. This indicates that the function has not been perfectly transferred.

Table 8: Performance of the transfer learning model, compared to the base model both with and without function altered CHF. Base data: 24461 datapoints (80% train, 20% test), Transfer data: 116 datapoints (24 train, 92 test)

Model	Data	NRMSE	Std	Mean
Base	Base	0.129	0.126	1.017
Base	Transfer	0.097	0.084	1.091
Base	No function	0.077	0.073	1.002
Transfered	Transfer	0.077	0.076	1.001

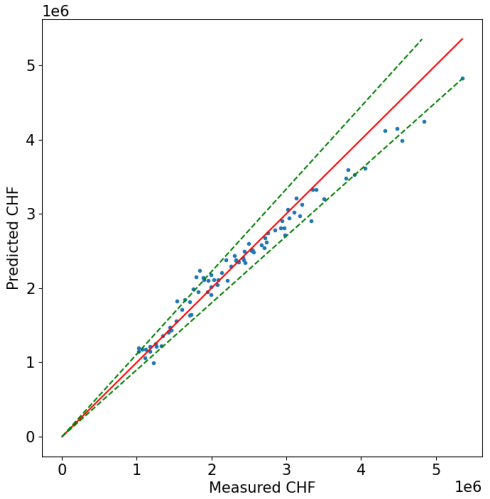
6.6.2 Transfer to Altered Geometry

The models used in transfer learning when using altered geometry are different from the main, optimal model described in Section 6.3. For the base model, the same hyperparameters are used as for the training of the main model. However, since the data is a subset only containing approximately a single diameter, the input only consists of four variables, instead of five. When transferring the data, the model has to adapt to another diameter, without knowing the alteration.

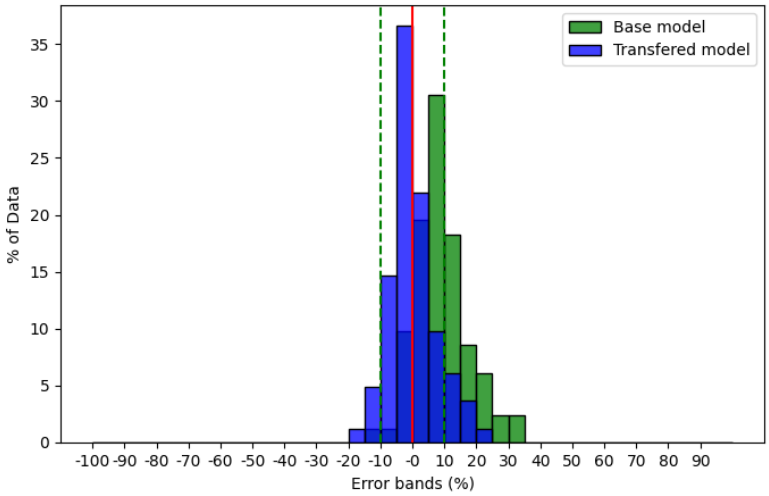
As we can see in Table 9, the base model is performing reasonably close to the complete model discussed in Section 6.3. However, when using the base model to predict the transferred data, we see that the results are unusable. The mean is far from one, and the NRMSE and Std have increased.

We attempt to transfer the base model. First, we freeze the base model and replace the base output layer with two layers of 40 neurons each and a new output layer, and then train the final layers. We see that when no fine-tuning is used, the mean is corrected, however, the NRMSE and Std have increased. The model is not transferring correctly.

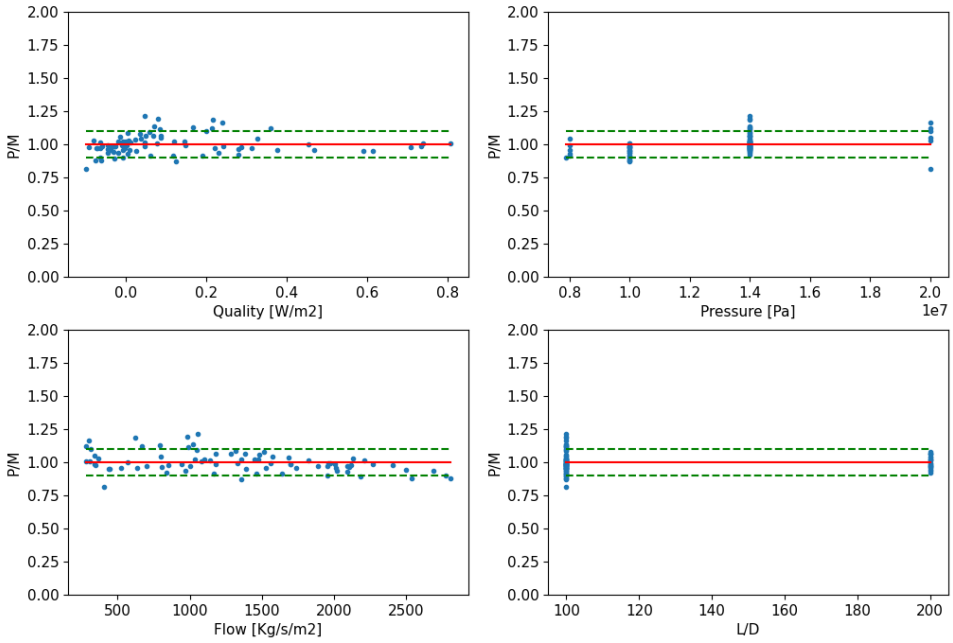
We thus introduce fine-tuning as an additional training step. Usually, fine-tuning is used on some layers at the end of the network. However, we see that even when using fine-tuning on the



(a)



(b)



(c)

Figure 14: Predictions of the transferred NN model when using the function altered CHF.

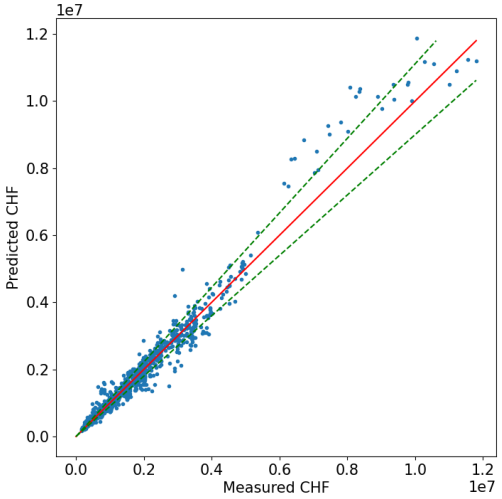
last 6 layers, there is virtually no difference in our network behavior. We thus test using fine-tuning on all our layers. Here we see that the NRMSE and Std both decrease. The predictions of this model can be seen in Figure 15.

To confirm that it is not the data we are transferring to that has a large variance, we also include the predictions of the main model: we see that the NRMSE and Std are very low, and we can thus discard this possibility.

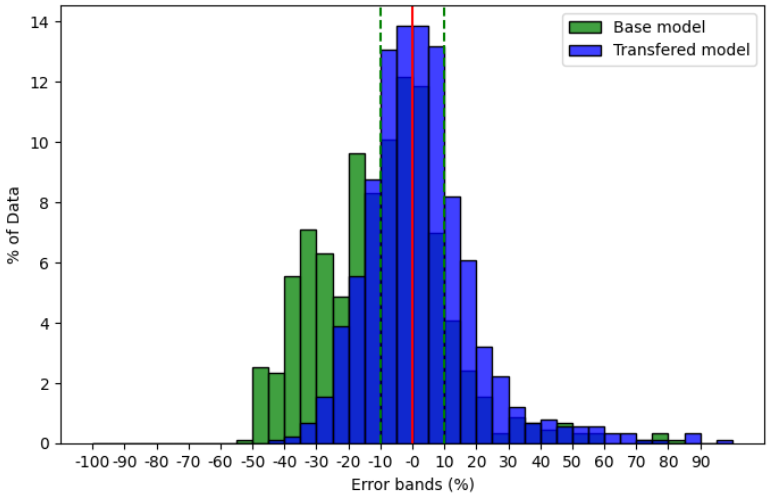
Table 9: Performance of the transfer learning model, compared to the base model and main model when using the geometrically altered CHF, and the influence of fine-tuning. Base data: 6053 data points (80% train, 20% test), Transfer data: 4521 data points (80% train, 20% test)

Model	Data	Fine tuning	NRMSE	Std	Mean
Base	Base	-	0.115	0.133	1.01
Base	Transfered	-	0.210	0.196	0.904
Transfered	Transfered	-	0.235	0.223	1.03
Transfered	Transfered	6 layers	0.237	0.242	1.011
Transfered	Transfered	All layers	0.186	0.182	1.028
Main	Transfered	-	0.103	0.086	1.000

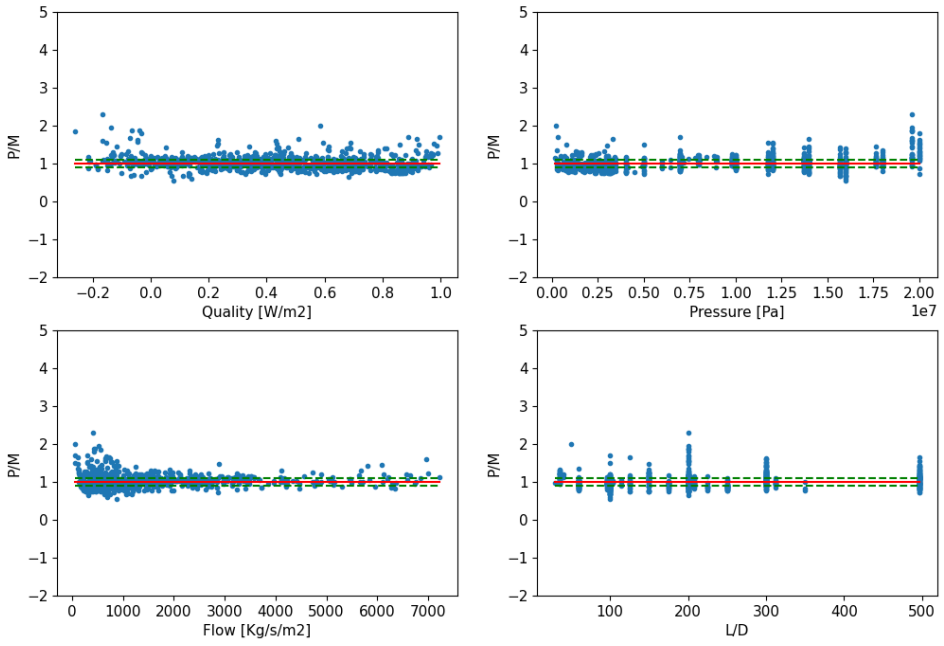
Let us now look more closely at our transfer model. we see in Figure 15a that the underestimations have improved by a lot, and the overestimations have worsened with the transfer. The error distribution has improved by a lot and is much smoother than for the base model. In Figure 15b we see that the errors are prevalent for both low and high measured CHF, however, the transferred model almost always overpredicts the CHF at high measured CHF. We see in Figure 15c that the error does not seem to be dependent on quality, pressure, or geometry, however, it seems to be larger at low flow. This might be because of the low flow being represented by only a few data points in the data the base model is trained on.



(a)



(b)



(c)

Figure 15: Predictions of the transferred NN model when using the geometrically altered CHF. The red lines represent perfect predictions, and the green lines represent 10% error.

7 Conclusions

The results in this thesis present adapted hyperparameter sets for our three prediction algorithms: ν-SVR, GPR, and NN. The results show that NN is the preferred model-building algorithm for predicting the occurrence of CHF. It outperforms both ν-SVR and GPR, and greatly improves on the drawbacks these methods have. NN does not predict negative CHF when the Measured CHF is very small, there are much fewer parameter-dependent errors, and the error distribution of the NN model is much more symmetrical. The optimal NN model also has significantly lower Std, and a more correct mean than the other methods.

All three algorithms show great improvements over the LUT, with the NN decreasing the Std of the predictions by almost a factor three, and the other models decreasing it by a factor two.

We see that the length parameter is an important parameter for predicting CHF, challenging the assumption made in the LUT. The NN model Std doubles when the length is removed as an input parameter. Even when removing the Length parameter, the NN outperforms the LUT significantly.

In the case where a model with fewer input variables, or dimensionless input variables are required, we conclude that the input variable combination G, P, $\frac{D}{L}$, X is performing well. This combination decreases the number of inputs to four, instead of five, and allows us to only use dimensionless geometric variables. Using only dimensionless input and output variables is not feasible with the methods utilized in this thesis. Using more dimensionless variables decreases the performance to an extent where the model no longer presents an improvement on the LUT.

The optimal model shown in the results is clearly improving the prediction trends for each parameter, except the quality. Our model has a similar structure to the LUT when predicting CHF with respect to the mass flux, pressure, and diameter, with differences in amplitude and displacement that are length-dependent. For the length, our model shows a clear decrease in CHF at low lengths, and a lesser influence at high lengths. For high qualities our model performs better than the LUT, however, predictions at $X = 1$ are questionable, and at low qualities our model also has unexpected behaviors. This is probably due to the small amount of data in this region.

The learning loss indicates that our optimal model is not seriously overfitting the data. The model behavior plots are smooth for all parameters except for low quality, and thus confirm the claim that our model is not overfitting, with the exception of low qualities, where the lack of data does not allow us to draw any conclusion on the overfit or underfit of the model.

The transfer learning tests performed had mitigated results. Transferring our model to a new database that has been altered with a function resulted in good predictions, however, the influence of the function did not disappear completely, as the mass flux parameter used in the function produces a small bias in the predictions. The resulting transfer was good and produces a similar amount of errors as when using the main model on the unaltered transfer data. When transferring to a different geometry we saw a large decrease in the prediction capabilities of the models. By using fine-tuning we managed to improve the transfered model, however, only when fine-tuning all the layers, meaning that the first layers in the base model are the problem. Since the first layers are generally seen as the ones deconstructing our problems into building blocks, this means our building blocks are wrong. Either our two databases are not representative of each other, or the transfer between geometries can not be performed in this way.

8 Future improvements

It is clear from our conclusions that the area where most improvements can be made are regarding transfer learning. The applications of transferring the model are highly practical, and are key to the application of this type of model in industry. To further test the possible application of transfer learning more tests are needed. Using the optimal model, instead of another similar model trained on a subset of the data, is required. To this end, additional databases are needed.

There are multiple interesting applications for transfer learning: the transfer to other geometries, for example, rod bundles, transfer to changes in the underlying physics, such as the usage of other coolants or the addition of flow mixers, and finally the transfer to non-uniform axial power

distributions. To further test these applications, the usage of databases representing physical measurements in these conditions is imperative. This will both allow the usage of the whole optimal model trained in this thesis as a base model, and the test of real-world scenarios of changes to the underlying physics, instead of theoretical changes that are as straightforward as a function.

The method of transfer learning will also need to be improved, as even the function transfer was not optimal. Optimization of the size and number of the transfer layer might lead to better results. Additionally, improvements to fine-tuning are certainly possible.

The model behavior at low qualities do not follow expectations, additional research into why might be necessary. Optimally, the model can be changed to be smoother at low qualities. For this, some type of Physics-informed method could be used, for example by adding a quality-dependent term to the loss function used in training, and in that way adding a constraint to the model.

Before the model can be used in production, an important requirement is local uncertainty estimations. A data-driven model used in production can not be as assertive in areas where there is no data, as where there are a large number of data points. For instance, the low quality regions' non-physical model behavior would be identified to yield large uncertainties. The advantage of GPR is that this uncertainty is part of the method. For the NN however this is not something that can be achieved naturally, and other methods need to be employed in addition. The most promising option to estimate local uncertainties is the usage of Ensemble methods [25], where multiple NN are trained using random initialization. The models can then be used to calculate a mean and standard deviation at a local point using the distribution of all the different model predictions. Another possible method to measure local uncertainties is the usage of Monte Carlo Sampling [26], where dropout is instead used during the evaluation to produce multiple outputs with the same model. This method is simpler to implement but is not as promising when it comes to predictions.

One specific uncertainty that could be of interest to improve the model is the disparity of uncertainty in the data. Since the data has been collected over a long time and with different measurement methods there could be a large disparity in the data uncertainties. The NN presented here has assumed that all the data have similar uncertainties, as we did not have sufficient information on the uncertainty disparities in data. If the disparity can be concluded in another way this could lead to a better model, since uncertain data would have less influence on the final model.

As we have discussed the length parameter is often seen as a parameter of lesser importance, as models for vertical round tubes that we use here often describe CHF as occurring first at the outlet of the tube. In the case of static heat flux, the length should therefore not impact the CHF. We see a large impact of the length parameter in our model which needs to be addressed. One possibility is that the length parameter describes one or multiple other mechanisms. For example, the length could describe how long it takes the model to react to the changes in inlet conditions. To further understand the mechanisms behind CHF investigating the meaning behind the length parameter, which we conclude to be important, could be of interest.

9 Acknowledgment

First I would like to express my gratitude to my supervisor Jean-Marie LeCorre: thank you for the tremendous help throughout the thesis, coming up with ideas, and the constant feedback on my work, thank you for all the help along the way. I would like to thank Gustav Robertson for guidance on machine learning and report writing. Thank you also Jerol Soibam for guidance on neural networks and Gaussian processes. Thank you all for the many interesting meetings we have had. I would like to thank my subject reader Alexander Medvedev for reviewing my report. Thank you also to Westinghouse electric Sweden AB for computational and financial assistance.

References

[1] B.-W. Yang, H. Anglart, B. Han, and A. Liu, "Progress in rod bundle chf in the past 40 years," *Nuclear Engineering and Design*, vol. 376, p. 111076, 2021.

[2] J.-M. Le Corre, *Flow regimes and mechanistic modeling of critical heat flux under subcooled flow boiling conditions*. PhD thesis, Mechanical Engineering, Carnegie Institute of Technology, Carnegie Mellon University, 2007.

[3] D. Groeneveld, J. Shan, A. Vasić, L. Leung, A. Durmayaz, J. Yang, S. Cheng, and A. Tanase, "The 2006 chf look-up table," *Nuclear Engineering and Design*, vol. 237, no. 15, pp. 1909–1922, 2007. NURETH-11.

[4] M. Gomez-Fernandez, K. Higley, A. Tokuhiro, K. Welter, W.-K. Wong, and H. Yang, "Status of research and development of learning-based approaches in nuclear science and engineering: A review," *Nuclear Engineering and Design*, vol. 359, p. 110479, 2020.

[5] J. R. Gurruchaga, "A comparison of machine learning algorithms applied to the problem of predicting the critical heat flux in a lwr,"

[6] X. Zhao, K. Shirvan, R. K. Salko, and F. Guo, "On the prediction of critical heat flux using a physics-informed machine learning-aided framework," *Applied Thermal Engineering*, vol. 164, p. 114540, 2020.

[7] H. Kim, J. Moon, D. Hong, E. Cha, and B. Yun, "Prediction of critical heat flux for narrow rectangular channels in a steady state condition using machine learning," *Nuclear Engineering and Technology*, vol. 53, no. 6, pp. 1796–1809, 2021.

[8] B. Jiang and F. Zhao, "Combination of support vector regression and artificial neural networks for prediction of critical heat flux," *International Journal of Heat and Mass Transfer*, vol. 62, pp. 481–494, 2013.

[9] M. He and Y. Lee, "Application of deep belief network for critical heat flux prediction on microstructure surfaces," *Nuclear Technology*, vol. 206, no. 2, pp. 358–374, 2020.

[10] M. He and Y. Lee, "Application of machine learning for prediction of critical heat flux: Support vector machine for data-driven chf look-up table construction based on sparingly distributed training data points," *Nuclear Engineering and Design*, vol. 338, pp. 189–198, 2018.

[11] B. Jiang, J. Zhou, X. Huang, and P. Wang, "Prediction of critical heat flux using gaussian process regression and ant colony optimization," *Annals of Nuclear Energy*, vol. 149, p. 107765, 2020.

[12] J. Soibam, A. Rabhi, I. Aslanidou, K. Kyprianidis, and R. B. Fdhila, "Prediction of the critical heat flux using parametric gaussian process regression," *Proceedings of the 15th International Conference on Heat Transfer, Fluid Mechanics andThermodynamics (HEFAT2021)*, pp. 1865–1870, 2021.

[13] D. Groeneveld, "Critical heat flux data used to generate the 2006 groeneveld lookup tables," *NUREG*, 2019.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah,

M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[16] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, "New Support Vector Algorithms," *Neural Computation*, vol. 12, pp. 1207–1245, 05 2000.

[17] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[18] F. Leibfried, V. Dutordoir, S. T. John, and N. Durrande, "A tutorial on sparse gaussian processes and variational inference," *CoRR*, vol. abs/2012.13962, 2020.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[20] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," *CoRR*, vol. abs/1512.03965, 2015.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[22] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[23] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems* (J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds.), vol. 24, Curran Associates, Inc., 2011.

[24] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 115–123, PMLR, 17–19 Jun 2013.

[25] S. Zhang, M. Liu, and J. Yan, "The diversified ensemble neural network," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 16001–16011, Curran Associates, Inc., 2020.

[26] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1050–1059, PMLR, 20–22 Jun 2016.