Evaluation of ML potential for modelling and process control of a reduction annealing process

Vidar Gimbringer, Björn Ziebeil

Master's thesis
2026:E6

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Abstract

While many industrial processes have been thoroughly exposed to machine learning models, the realm of powder metallurgy is still underexplored. This thesis aims to investigate the potential for a machine learning model to predict the characteristics of metal powders after a reduction annealing process using time series process data from the furnace. A supervised learning model was developed to predict six target variables: three particle size distributions, two chemical composition properties, and a density. The pipeline involved filtering and preprocessing data, model training, and performance evaluation.

Multiple families of machine learning methods were explored and implemented, including linear methods, tree-based ensemble methods, and sequence models. Hyperparameter values were systematically tuned from a range of values to find the best configuration; each model's performance was then assessed using cross-validated error metrics. Using SHAP analysis, it was possible to identify the most prevalent and important features for predicting each output for each model.

Results show that performance varies for each target variable, and while no model was completely dominant, the most successful predictions were made by LSTM, XGboost, and the Elastic Net, with all models presenting some overlapping features with the highest SHAP scores. The findings show that all three types of models tested were applicable and offer promising potential for the further rollout of machine learning in this field.

Contents

Chapter 1

Introduction and Purpose

During an industrial process, most modern facilities will closely monitor a product's progress throughout its production. At Höganäs AB, the world's largest producer of metal powders, a customer orders a specific and often unique composition of metal powder tailored to their use. A vital step in the powder creation is the reduction annealing process, which softens the powder through controlled heating. To fulfill the demands of multiple products, Höganäs uses a furnace with changeable settings, allowing for different metal powders to be created within the same chain of industrial steps, streamlining the process. Close measuring and control gives a clear insight into the values of features such as furnace temperature, cool water temperature, and fan speed. In order to assure customers a high-quality and consistent product, an analysis is done regularly after the process to make sure properties such as particle size, density, and chemical concentrations are within the correct span. A batch of metal typically takes around 24 hours to complete its annealing process and for its samples to be analysed. Any faults in the product are detected, and alterations to the furnace features are made.

The next step in optimising this production line is to remove the 24-hour delay needed to measure the product's quality. An efficient assisting tool to complement this system would be the ability to predict and forecast the product's quality during the process. With this, tweaks to parameters can be made in real time, which saves time and money. Large amounts of data are available from labelled powder batches sent through the furnace, along with corresponding analysis data measuring the final product's properties. This makes the reduction annealing process an ideal trial for implementing and investigating the potential of machine learning for Höganäs. Looking at the data from the furnace from a metal powder, the goal with this project is to train a machine learning model, by feeding it the input from the furnace with its matching analysis output. Learning from large amounts of data, the model can find hidden correlations and be able to make predictions of the analysis properties for a batch of metal, given its furnace data

during the process.

Our purpose is to investigate different types of machine learning techniques and evaluate their performances. Based on the data available, we will strive to create a robust model that can make accurate predictions and path the way for machine learning to be expanded and implemented into other industrial processes at Höganäs.

Chapter 2

Background

This chapter aims to introduce and provide sufficient knowledge to fully understand the production process on which this thesis is based on. The chapter starts by introducing the concept of powder metallurgy and some relevant processes leading up to the production of the metal powders examined in this thesis. Lastly, the actual annealing process and its purpose and effect in the production line is explained.

2.1 What is Powder Metallurgy?

Powder Metallurgy (PM) is the branch of metallurgy that deals with the manufacture of metal powders, and objects/materials that are produced by metal powders [1]. The PM process has many advantages compared to other conventional metal forming technologies such as forging, metal casting or machining. The powder technology provides unique opportunities to manufacture materials tailored to specific material properties and microstructures as well as complex geometric dimensions. In contrast to casting or wrought processing, PM decouples melting from shaping: powders are first manufactured (e.g., by atomization), then consolidated by routes such as press–sinter, metal injection moulding (MIM), hot isostatic pressing (HIP), or additive manufacturing (AM). Among these, *pressing and sintering* dominates high-volume ferrous structural parts due to excellent material utilization, dimensional control, and the ability to engineer porosity and composition at low cost. Early sintering of metallic powders dates back more than a century, but industrial PM matured in the early–mid 20th century with reliable iron powder production and continuous sintering furnaces. Sweden, and Höganäs AB in particular, played a central role, scaling sponge-iron and later water-atomized iron powders for the rapidly growing press–sinter market. Continuous improvements in powder production, lubricants, tooling, and furnace atmospheres enabled today's

high-volume, tight-tolerance ferrous PM industry. The applications for sintered products is wide and includes products such as gears, pulleys and structural components for the automotive and aviation industry. In this thesis we will be looking closer at the production process of two commonly used low-alloy iron-based powders used for pressing/sintering, *Astaloy CrM* and *Astaloy CrA*. More specifically, we will be focusing on the *Reduction Annealing Process* (see 2.1.1) during Astaloy production.

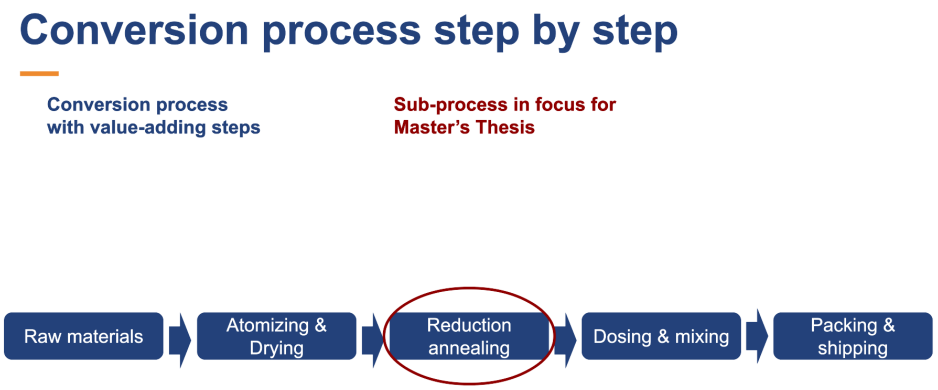


Figure 2.1: Step-by-step process for the production of Astaloy powders

Low-Alloy Powders

Astaloy CrM is a chromium (Cr) and molybdenum (Mo) steel powder (approx. 3 % Cr, 0.5 % Mo) designed for high hardenability and strength after conventional sintering, with additional gains under sinter-hardening or high-temperature sintering. In production it offers good dimensional stability and a robust path to high performance in press–sintered structural parts [2].

Astaloy CrA is a cost-effective Cr-only steel powder (approx. 1.8 % Cr) intended for medium-to-high strength applications; alloyed only with carbon it provides a fine pearlitic or sinter-hardened microstructure, attractive wear resistance (e.g. after nitriding), and stable, easy processing on standard press–sinter lines [3] [4].

Atomization

The atomization step is where the raw material (often scrapped car parts) are melted and disintegrated into small droplets, which consolidate into a powder of particles with a wide range of size distribution and high specific surface area after

rapid quenching. For low-alloy steel powders used in press–sinter, *Water Atomization* is the primary route, which uses high pressure water jets to disintegrate the stream of molten steel. Rapid quenching and contact with water promote thin surface oxide films (Fe–O and alloy-oxide mixtures), which are later addressed by thermal treatment [1]. Post-atomization steps include drying, magnetic separation, screening/classification, and tempering or annealing to adjust hardness and ductility.

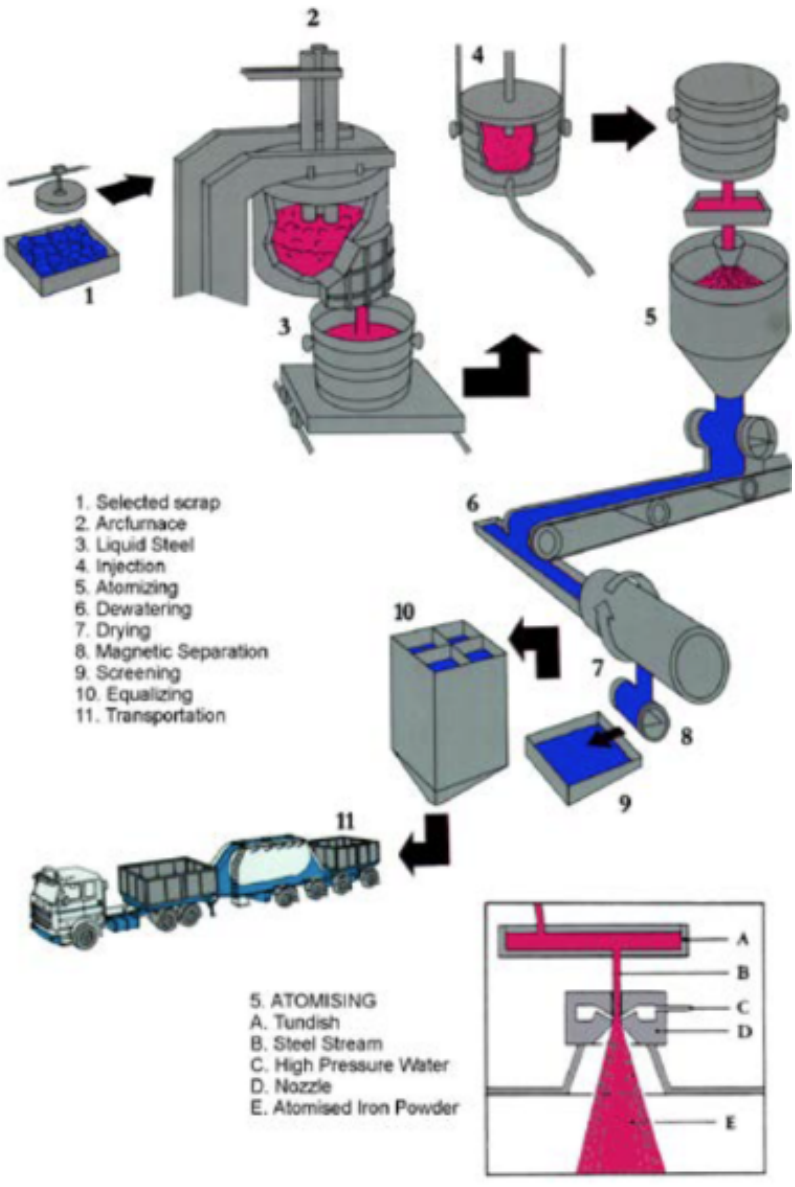


Figure 2.2: Schematics of Production Process from scrap to Powder

2.1.1 The Reduction Annealing Process

Reduction annealing is the upstream heat-treatment that "resets" water-atomized iron and low-alloy steel powders before pressing and sintering. Its primary functions are to (i) remove or transform surface oxides formed during atomization and drying, (ii) stabilize surface chemistry (oxygen and carbon), and (iii) restore ductility in work-hardened particles to improve compressibility for reproducible sintering. In practice, the powder is conveyed through a continuous furnace with preheat, reduction/soak, and controlled-cooling zones (see fig. 2.1) under a reducing atmosphere, often hydrogen gas, H_2.

The chemistry is governed by well-known equilibria between metal/oxides and the furnace gas. For iron oxides, the key reaction is

$$FeO + H_2 \rightleftharpoons Fe + H_2O,$$

so the ratio p_{H_2O}/p_{H_2} (measurable as dew point) sets the oxygen potential, at which oxide reduction proceeds. Below the equilibrium ratio the process pulls oxygen of the powder surface and the oxide reduces, above it the metal re-oxidizes. Ellingham-type relations and Fe–gas equilibrium diagrams are routinely used to select temperature–dew-point windows; a practical implication is that water vapour that is harmless at the hot zone can be oxidizing in preheat and cooling sections, making low dew points (low water content in the atmosphere) essential throughout the profile. Alloy oxides are more refractory than FeO, so chromium- and manganese-bearing films common on Astaloy-type powders require either drier atmospheres, higher soak temperatures, or longer residence times to achieve comparable reduction. For example, chromium can remain oxidized up to high temperatures unless the p_{H_2O}/p_{H_2} ratio is very low, which explains why insufficiently reducing conditions lead to poor compressibility and sluggish sintering despite adequate thermal input [5]. Industrial lines therefore couple temperature control with continuous atmosphere monitoring. Dew point (oxygen potential) and carbon potential are tracked and adjusted via gas composition and flow with the objective being to maintain reducing conditions in the hot zone, avoid re-oxidation during cooling, and prevent side reactions at lower temperatures.

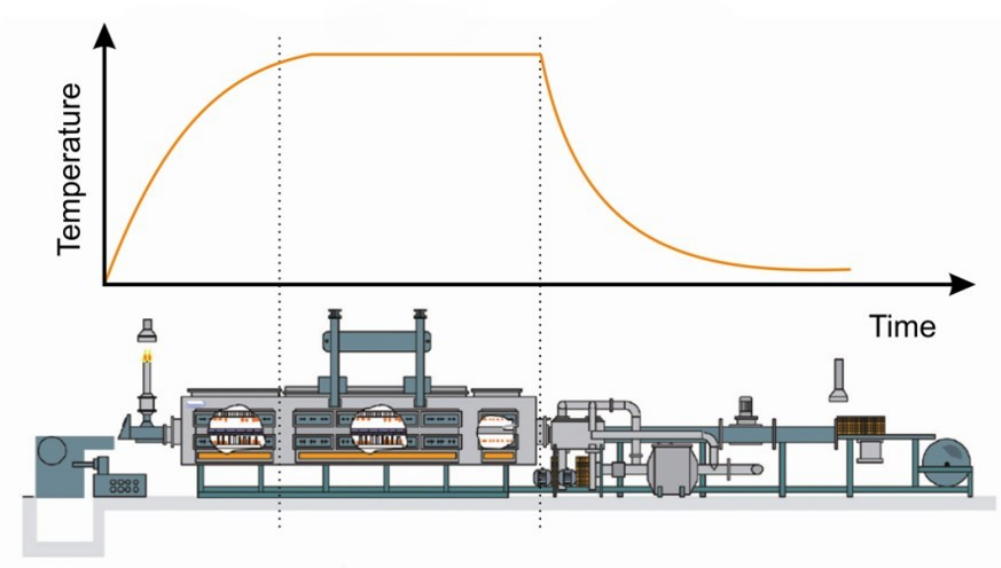


Figure 2.3: Furnace Process

2.2 Related Work

A lot of work has previously been done in the realm of machine learning (ML) for industrial processes [6, 7, 8], some even on the sintering furnace [9]. The purpose of the implementation of machine learning can be grouped into four main practical roles; (i) *soft sensing* —estimating difficult or delayed laboratory variables from readily available signals; (ii) *forecasting and control support* —predicting near-term process trajectories to improve set-point plans; (iii) *fault detection and diagnosis* (FDD) —identifying incipient anomalies to prevent scrap or downtime; and (iv) *recipe/window optimization* —navigating multivariate trade-offs among quality, energy, and throughput. Reviews from the process-systems community and Industry 4.0 literature underline that adoption succeeds when models are aligned with plant data realities (heterogeneous sensors, small data per unit, covariate shift), and when outputs are operationally interpretable for engineers. Tree ensembles and kernel methods remain strong baselines on tabular plant data; gated RNNs (GRU/LSTM), 1D CNNs, and Transformers become advantageous when temporal structure and long-range dependencies dominate [10].

Soft Sensors and Quality Predictions

AI-driven soft sensors compress multi-sensor streams into estimates of unmeasured (or slowly measured) variables—exactly the setting in which laboratory oxygen,

carbon, or green density are predicted from furnace and section-level tags. A recent survey highlights robust pipelines (data cleaning, drift handling, uncertainty, and model maintenance) and documents that ensembles and deep temporal models consistently outperform static regressors when dynamics matter. These works emphasize dew-point/oxygen-potential surrogates and hybrid features that combine physics with data [11]. In powder metallurgy specifically, gradient boosting, AdaBoost, and random forests are used to estimate mass and length of sintered workpieces from press and environment signals, reporting sub-percent RMSE on a production dataset. Thus indicating that carefully engineered features plus nonlinear ensembles can deliver plant-grade accuracy on modest data volumes [12].

Time-series Forecasting for Thermal Processes

Thermal lines (reheating and sintering furnaces, continuous annealing) exhibit delays, recirculation, and slowly evolving boundary conditions and are thus suitable for sequence models. GRU-based predictors have been shown to forecast internal temperature from fuel/air/temperature time series and to integrate with feed-forward control to reduce overshoot and oscillations, illustrating the value of gated recurrent units when long-range dependencies are present but data are limited. These application studies motivate our selection of GRU/LSTM as candidates for quality prediction from section-wise PV streams [13] [14].

FDD and Operational Reliability

Fault detection and diagnosis (FDD) in chemical and steel processes provide a taxonomy spanning quantitative model-based, qualitative knowledge-based, and data-driven approaches. Recommendations use residual analysis with statistical decision rules, combine multiple models, and favor interpretable indicators map directly onto PM lines where early detection of atmosphere excursions (dew-point spikes, oxygen leaks) or cooling-zone upsets is critical to avoid irrecoverable quality loss. These reviews also stress lifecycle management (retraining under drift, alarm rationalization), which we adopt in our evaluation protocol [15, 16, 17].

Optimization and Physics-Guided Modeling

Beyond prediction, ML is used to optimize grade/quality or energy under constraints. In extractive and pyrometallurgical contexts, stacked ensemble learners (RF/GB/SVR with linear meta-models) have been applied to predict and optimize oxide grades (e.g., TiO_2 in high-Ti slag), illustrating sample-efficient surrogate optimization on real plant data. For metallic materials design and forming, hybrid strategies that embed physical metallurgical principles (phase/strength models, diffusion surrogates) into data-driven learners improve generalization and engineer

trust. This approach may be attractive for reduction annealing where thermodynamic constraints (oxygen potential, carburization) are well understood [18] [19].

Chapter 3

Theory and Definitions

In this chapter the underlying ML theory is presented together with some background theory regarding the models used during the thesis.

3.1 Machine Learning Fundamentals

Machine learning (ML) addresses problems where we cannot (or do not want to) hard-code the mapping from inputs to outputs. Given an input space $\mathcal{X}$ (features) and an output space $\mathcal{Y}$ (targets), the goal is to learn a function $f_\theta : \mathcal{X} \to \mathcal{Y}$ that makes accurate predictions while generalizing well to unseen data. Formally, for a loss $\mathcal{L}$ and joint distribution $P(X, Y)$,

$$\theta^\star \;=\; \arg\min_\theta \; \mathbb{E}_{(X,Y)\sim P}\Big[\mathcal{L}\Big(Y, f_\theta(X)\Big)\Big],$$

which we approximate using a finite sample $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ drawn from historical data [20, 21].

Traditional programming vs. ML.

In traditional programming, a human writes explicit rules that map inputs to outputs. In ML, we specify a *model class* (linear models, trees/ensembles, neural networks), a *loss*, and an *optimizer*; the rules are inferred from data. This is especially valuable in industrial contexts (e.g., thermal processes) where relationships are nonlinear, time-varying, and partially observed.

Core components.

Data are pairs $(\mathbf{x}_i, y_i)$; *features* are the measured or engineered descriptors in $\mathbf{x}_i$ (e.g., lags, rolling statistics, rates of change, exposure integrals); the *target* y_i is the quality or state we want to predict; the *model* f_θ produces a *prediction* $\hat{y}_i = f_\theta(\mathbf{x}_i)$.

- Features: x_i - input parameters
- Target: y - the value we want to predict
- Prediction: $\hat{y}$ - value predicted by model
- Hyperparameter: λ - parameter that impacts the model with a value we can tune
- Amount of samples: n
- Amount of features/predictors: p

Model parameters vs. hyperparameters.

Parameters θ are learned from data (weights in a neural network, split thresholds and leaf values in a tree, coefficients in a linear model). *Hyperparameters* λ control model capacity and training behaviour (e.g., regularization strengths, tree depth, learning rate, convolutional window sizes). Parameters are fit by minimizing the training loss; hyperparameters are chosen by validation (see Section 3.1.1).

Overfitting, underfitting, and bias–variance.

Let $f^*(\mathbf{x}) = \mathbb{E}[Y \mid X = \mathbf{x}]$ be the optimal regression function. For squared loss and a fixed $\mathbf{x}$, the expected prediction error decomposes as

$$\mathbb{E}\Big[(Y - \hat{f}(\mathbf{x}))^2 \mid X = \mathbf{x}\Big] = \underbrace{\Big(f^*(\mathbf{x}) - \mathbb{E}[\hat{f}(\mathbf{x})]\Big)^2}_{\text{bias}^2} + \underbrace{\mathbb{V}\text{ar}[\hat{f}(\mathbf{x})]}_{\text{variance}} + \underbrace{\mathbb{V}\text{ar}[Y \mid X = \mathbf{x}]}_{\text{irreducible noise}},$$

where $\hat{f}$ is the learned predictor viewed as a random object due to sampling variability. (Derivation: add and subtract $\mathbb{E}[\hat{f}(\mathbf{x})]$, expand the square, use law of total expectation.) Underfitting corresponds to high bias (model too rigid); overfitting to high variance (model too flexible). The aim is to balance both, given the noise level in the labels [21, 20, 22].

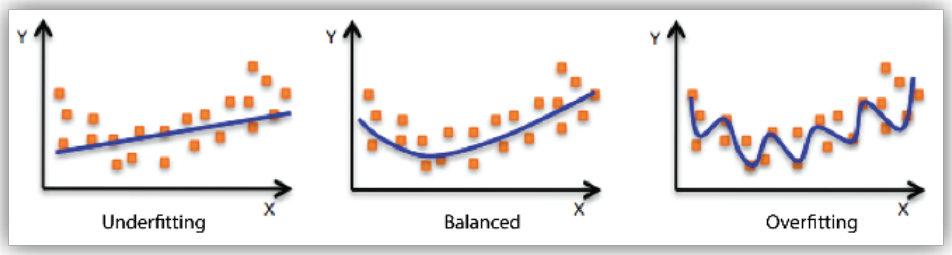


Figure 3.1: Underfitting and overfitting visualized

3.1.1 Model Training and Evaluation

Training consists of (i) defining an *empirical risk* on the training sample, (ii) minimizing it to estimate θ, and (iii) using properly separated *validation/testing* periods to choose hyperparameters and estimate generalization. In time-dependent settings, all splits must be sorted according to their time stamps to avoid *leakage*.

Loss functions and Performance Metrics

For regression targets (e.g., quality variables after annealing), the evaluation metrics used are:

$$\text{Mean Absolute Error (MAE)} = \frac{1}{n}\sum_i |y_i - \hat{y}_i|,$$

$$\text{Mean Squared Error (MSE)} = \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2,$$

$$\text{Root Mean Squared Error (RMSE)} = \sqrt{\text{MSE}},$$

$$\text{Normalized Mean Absolute Error (nMAE)} = \frac{\frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|}{\max(y) - \min(y)},$$

$$\text{Normalized Root Mean Sqaured Error (nRMSE)} = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}}{\max(y) - \min(y)},$$

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}, \quad \text{where } \bar{y} \text{ is the mean of values: } \bar{y} = \frac{1}{n}\sum_i y_i,$$

$$\text{Mean Absolute Percentage Error (MAPE)} = \frac{100}{n}\sum_i \left|\frac{y_i - \hat{y}_i}{y_i}\right| \quad (y_i \neq 0),$$

$$\text{Symmetric Mean Absolute Percentage Error (SMAPE)} = \frac{100}{n}\sum_i \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2}.$$

MAE is in physical units and robust to outliers; RMSE emphasizes large deviations. For outputs with different magnitudes, the MAE and RMSE scores will be dominated by outputs with larger values. The correct way to compare outputs is to normalize these scores based on the output's range of values, which are the nMAE and nRMSE scores. The R^2 metric is a *relative* goodness-of-fit and can its score range from $[-\infty, 1]$. Where a score of 1 is a perfect match, while a negative score means the model's prediction is worse than the mean, anything over 0 indicates the model finds trends in the data. The R^2 scores are intuitive but are

unstable for an output with low variance, as this results in a small denominator [21]. MAPE is undefined when $y_i = 0$ and unstable near zero; SMAPE mitigates this issue, but is still sensitive to tiny denominators i.e., outputs with very small values [23].

SHAP

When training a model with many input features, determining which of all features are the most important to the models prediction is essential. Understanding which features are influential allows one to simplify the model by removing irrelevant features and strengethen the ones that matter most. In this report, we use SHAP scores to determine which features are dominant and improve interpretability of our results.

SHAP or SHapley Additive exPlanation is an application of Shapley scores from game theory to machine learning models [24]. Each feature receives a score based on how much it contributes to the models prediction. This score is calculated by comparing the models output when the feature is included and excluded.

The full formula for a Shapley value is,

$$\phi_i(f, x) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(|N| - |S| - 1)!}{|N|!} \left[f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S) \right]. \tag{3.1}$$

The terminology is as follows: the full set of features is N, where $S \subseteq N \setminus \{i\}$ is any subset of features not containing feature i. The models prediction at input x, using the features found in the subset S is $f_S(x_S)$. The Shapley value for feature i at input x is $\phi_i(f, x)$ and determines the features average contribution for all possible subsets. The weighting $\frac{|S|!\,(|N|-|S|-1)!}{|N|!}$ assures that the algorithm assigns a fair contribution proportional to the amount of subsets [25]. Sorting all features by their SHAP score gives us a straightforward result of the most important features.

3.1.2 Types of Machine Learning

Machine learning models can be divided into four separate types, with each having its own area where they excel.

Supervised Learning

Supervised learning is the act of training a model for a problem in which we already know the correct answer. Supervised learning can be divided into classification and regression problems [21]. A classification problem involves labeling an input into a class, predefined by the user. Typical examples are to identify what class an

image or text belongs to, such as what animal the image depicts or decide whether an email is spam or not.

Regression involves predicting a continuous stream of numerical values based on input data and its connection to the corresponding output. The input data contains multiple variables, known as features or predictors and the output data are the target variables, which the model is learning to predict. A regression model aims to correctly predict the output using only the input. Examples of regression problems may be the selling price of apartments based on features such as square meters, location, date of sale, etc, or the physical properties of a metal after undergoing an industrial process with features such as temperature, pressure, and weight, as done in this thesis.

A typical method for training a supervised learning model is to split the data into training data and test data. The training data is fed into the model with both inputs and outputs, which allows the model to learn and improve. The testing data is withheld from the model, and is used to validate how it performs. Input data from the testing set is given to the model, and the predicted output is compared to the testing data output [26]. The goal of the model is to minimize a loss function, which evaluates how accurate the model is, a typical loss function is mean-squared error although many others are viable, as mentioned in 3.1.1. The internal parameters of the model are optimized to minimize the loss function using optimization algorithms and performance metrics that are discussed in later sections.

Unsupervised Learning

The alternative case where we do not have a defined correct output is known as unsupervised learning. For these problems, the data is unlabeled, meaning there are no predetermined connections or corresponding output values. The model instead searches for hidden patterns and structures in the data without human guidance. Grouping together similar data points is known as clustering, with one of the most commonly used algorithms being K-means clustering. This algorithm groups the data into K distinct spherical groups based on their Euclidean distances from one another. This can be applied to any dataset and, when visualized, provides a clear view of how the data is structured [21].

For a dataset containing many features it is often beneficial to perform a dimensionality reduction to reduce the complexity. Principal component analysis (PCA) is a technique that linearly transforms the data to a new coordinate system in order to find the directions for which the data has the greatest variance; these directions are known as the principal components [27], PCA and how it is applied in this thesis is more thoroughly discussed in Section 6.5.

Unsupervised learning models are often applied in cases where we wish to

extract the key features to a problem, visualize the data or detect outliers and anomalies. A typical case could be customer segmentation, where the goal is to determine behavioural patterns without predefined labels based on store data.

Reinforcement Learning

Reinforcement learning is the approach of training a model through rewards and feedback. An agent makes decisions and learns by trial and error, with every action being penalised or rewarded immediately or with a delay, based on the environment's setup. For every state the agent encounters, the available options are different. The agent aims to find an optimal policy, which is the strategy for taking the best action in each state. Through repeated attempts, the agent learns a policy that maximizes its rewards over time. For every state and action pair, a value function estimates the expected future return.

The optimal policy is unique to each task, and requires balancing both exploration and exploitation. Exploration involves trying new strategies and choices, that may be more risky but could yield higher rewards. Exploitation involves utilizing already proven strategies from previous runs. Thus, the rewards given to the agent, the reward function, is crucial in designing an efficient reinforcement learning model [28].

For an agent learning to play the game Snake, where the player controls a snake that aims to pick up as many apples in a grid as possible, without steering into a wall. It is straightforward to assign picking up apples as a positive reward, while dying is punished with negative feedback. The balance of how these two rewards counter each other is more complex; if the dying penalty is extremely high the agent will focus on staying alive and may never achieve the objective of completing the game. If picking up apples is given a significantly higher reward than dying, the agent may be reckless, picking up many apples but dying quickly.

Reinforcement learning mimics how humans learn new tasks through constant feedback from our environment and is commonly used in fields with repetitive tasks such as games, robotics, and autonomous control.

Generative AI

The last subtype of the four different machine learning models is generative AI, which learns from data and is capable of generating new, original content. By processing vast amounts of data, the models uncover underlying complex patterns and identify the most relevant features. The learning process is a type of unsupervised learning where the models try to predict the next output based on what came before it. The deep learning architecture that powers generative AI models relies on neural networks with many layers, which enables them to generate an

output based on a given input or prompt [29].

The key to a high-quality generative AI lies in the quality of its training data. If the training data contains flaws or biases, the model may generate skewed and misleading results, with high confidence; this mismatch is known as hallucination. Generative AI models can be used to create chatbots which are known as Large Language Models (LMM) such as ChatGPT, but the applications also extend to image generation, music creation and writing computer code. Due to the extensive amounts of data needed, generative AI is among the most resource-intensive types of machine learning [30].

3.1.3 Gradient Descent

A task that often shows up in machine learning optimization is to find a method to minimize a function - typically the cost or loss function. The most commonly used solution is the gradient descent algorithm. For a differentiable function the gradient ∇f is the vector containing all partial derivatives and points in the direction of steepest ascent [31].

$$\text{Gradient for a function of three variables: } \nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right).$$

As the gradient points in the direction of steepest ascent, the negative gradient will show us the direction of steepest descent, which will lead to a trajectory of finding the minimum. For a multi-variable function f that is defined and differentiable in the vicinity of a starting point $\mathbf{x_0}$, taking a step in the opposite direction of the gradient gives us the next point $\mathbf{x_1}$. How far one wants to step in this direction before computing the gradient again is determined by the learning rate η [32], defined as,

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta \nabla f(\mathbf{x}_n). \tag{3.2}$$

For an ideal function that is differentiable, convex and smooth, a sufficiently small step size will result in $f(\mathbf{x_0}) \geq f(\mathbf{x_1}) \geq f(\mathbf{x_2})...$ and the algorithm will generate a sequence of minimizing points that will bring us closer to the minimizer for each iteration. The algorithm stops when the termination condition is met, such as when the computed function values between iterations is very small, or the algorithm may be capped by a maximum number of iterations to save processing power. Selecting a learning rate η that ensures a stable and fast convergence is a challenge that varies for each function.

A common issue for the gradient descent involves the existence of a local minimum or a saddle point, where the gradient is near zero, which forces the algorithm to take many small steps. In such regions, the algorithm may be very slow or converge to a suboptimal point. A variant known as stochastic gradient descent

(SGD) [32], uses smaller batches of data near the point for each calculation of the gradient and is thus, more efficient for larger datasets. SGD with momentum [33], is a tweaked SGD algorithm that memorizes gradients from previous points and takes these into account when computing the next point. This "momentum" from the previous points can help the algorithm to escape from local minima and saddle points. SGD's efficiency for data-heavy tasks makes it a regularly used version of gradient descent for machine learning problems.

3.1.4 Neural Networks

Another core algorithm within machine learning is a neural network, which is modeled to mimic the human brain. The network consists of layers of linked nodes, which represent neurons. Data is passed from the first layer, known as the input layer, through to the last layer, the output layer. Between the input and output layer there may exist one or more hidden layers that add to the complexity of the network. A network with at least two hidden layers is known as a deep neural network [34].

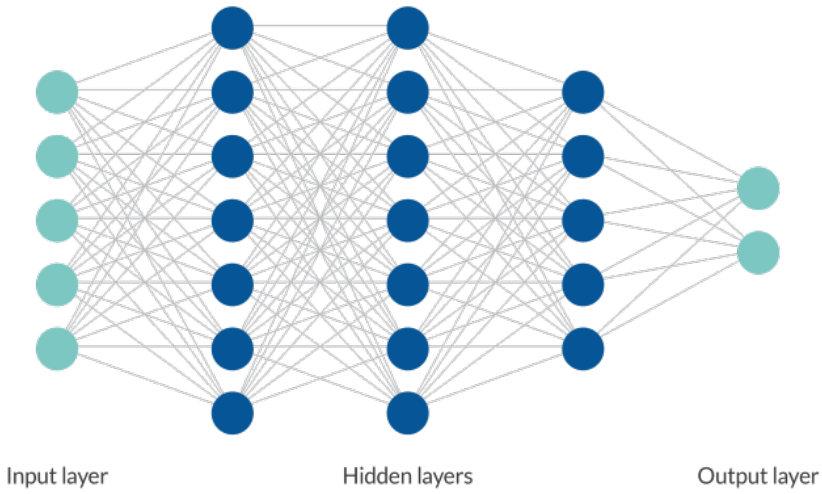


Figure 3.2: A neural network consisting of connected nodes and layers

The links between the nodes, also called edges, have an assigned weight, which determines the importance of the input passing through. Initially, these are typically set to small random numbers to avoid symmetry and are updated during training. In addition to the link weights, each node in the hidden and output layers can have a bias. The inputs from the connecting nodes and the weights of the connecting links are taken into account, and a resulting weighted sum is calculated. An activation function processes the weighted sum before sending it forward to the next

node. The activation function controls how much of the neurons' signal should be sent to the next layer. The selection of an activation function varies; but the key idea is that a nonlinear activation function enables the neural network to process nonlinear data, enabling the network to learn far more abstract features and patterns. For a node, with bias b, n connecting links with weights w_i and inputs x_i, the corresponding output of the node is processed by the activation function f,

$$f(b + \sum_{i=1}^{n} x_i w_i). \tag{3.3}$$

Without an activation function, each layer simply becomes a linear transformation, and as the sum of two linear combinations are also a linear combination, the entire network collapses down to a single linear equation [35]. Examples of activation functions commonly used are the sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$, which computes a number from 0 to 1, or the ReLU function: $f(x) = \max(0, x)$, which only passes forward a value if it is positive. The method of updating the weights for every link is through the use of the backpropagation algorithm.

Backpropagation

The backpropagation algorithm is the standard method used to train and optimize neural network's internal parameters, and involves minimizing the error of the network's prediction. A forward pass, which is the network output from any given signal is sent forward through the layers and the resulting output, is compared to the ground truth value with a loss function. This is followed up with a backwards pass, which sends the loss function back through the layers in the network.
For each node, the derivative of the loss function is calculated, the weight and bias for this specific neuron impact the value. Using the gradient-descent algorithm, the weights and biases are updated through each iteration in order to minimize the error [36]. The formula for updating the current weight W_i to a new weight W_i^*, with an error E, defined by a loss function such as the squared error $(y - \hat{y})^2$, is shown below,

$$W_i^* = W_i - \eta \frac{\partial E}{\partial W_i}. \tag{3.4}$$

The learning rate η, is the hyperparameter that decides how aggressive the changes to the weights and bias are and needs to be tuned to an optimal value. Through the use of repeating backpropagations, the neural network performance keeps improving.

3.2 Linear Models

This section introduces the family of linear machine learning models, starting with the linear regression. We start from ordinary least squares as our loss function, and then add regularization in the form of LASSO regression (ℓ_1 regularization) and Ridge regression (ℓ_2 regularization), and finally the Elastic Net (ℓ_1+ℓ_2), which is a combination of both. The following subsections define each model, objective, and training setup used in our experiments.

3.2.1 Linear Regression

Given a supervised learning problem where we have an input and its corresponding output, the most basic model is the standard linear regression. For a simple problem consisting of only one variable x and a single output $\hat{y}$, the linear regression is represented by the equation $y = mx+c$. This is essentially the task of identifying the best-matching straight line fitted to the data. For multivariable inputs, $X = (x_1, x_2..., x_p)$ the corresponding output is dependent on many inputs,

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + \epsilon \tag{3.5}$$

or in matrix form $\mathbf{y} = \mathbf{X}\beta + \epsilon$, and we have a multiple linear regression. The residual ϵ is assumed to have zero mean. For higher-dimensional problems y may be a vector of outputs, and linear regression becomes fitting a hyperplane to the points instead of a line. The hyperplane is chosen by minimizing the error between the points and the plane, and the by far most popular method is least squares, where we pick coefficients β_p to minimize the residual sum of squares (RSS) between the models prediction $\hat{y}$ and the actual values y [21],

$$RSS(\beta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - x_i^T\beta)^2. \tag{3.6}$$

The objective is a convex quadratic function, which guarantees the existence of a minimum. Switching to matrix notations we can express the solution, by differentiating with respect to β.

$$RSS(\beta) = \sum_{i=1}^{n}(y_i - x_i^T\beta)^2 = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta), \iff$$

$$\frac{\partial}{\partial\beta}RSS(\beta) = \mathbf{X^T}(\mathbf{y} - \mathbf{X}\beta) = 0 \iff \hat{\beta} = (\mathbf{X^T X})^{-1}\mathbf{X^T y}.$$

For the case of $\mathbf{X^T X}$ being invertible, this is the unique solution that produces the optimal coefficient selections $\hat{\beta}$. A prediction the linear model makes for input i

is, $\hat{y}_i = \hat{y}(x_i) = x_i^T \hat{\beta}$. While linear regression is a foundation of machine learning and is simple to apply to a dataset, it has many limitations. The model assumes a linear and independent relation between the input features x and the output target y, which is rarely the case. A commonly encountered issue is multicollinearity, which occurs when two variables are closely correlated, making it hard for the model to determine the individual effects. The probability of correlation, even by chance, between features increases with the amount of features, which is a result of models behaving worse as the dimensions increase, this is known as the curse of dimensionality [37]. Solutions to this are techniques such as ℓ_1 and ℓ_2 regression, which limit their impact, or principal component analysis (PCA) which reduces the dimension. Linear regression is a starting point in machine learning and is improved upon by many other models.

3.2.2 LASSO regression / ℓ_1 - Regularization

When a machine-learning task involves a large number of features it is logical to assume in some cases, that many of these features may have minimal impact on the model. ℓ_1-Regularization solves this by performing feature selection and encouraging a sparse model, forcing the model to identify only the most relevant data predictors. As the name suggests, ℓ_1 refers to the absolute value of the coefficients. ℓ_1-Regularization adds a penalty term to the cost function for every coefficient proportionate to its absolute value, which is the ℓ_1 norm. When features are highly correlated, the columns in the matrix $\mathbf{X}$ are almost linearly dependent. This results in the matrix $\mathbf{X^T X}$ becoming near singular, which causes large variances in coefficient estimations and large magnitudes in the ordinary least squares regression. This issue is mitigated by the LASSO, which solves a constraint optimisation problem instead of inverting $\mathbf{X^T X}$.

For a set of highly correlated features, ℓ_1-regularization selects one of these features arbitrarily to keep and shrinks all others towards zero. When a coefficient of a feature reaches zero, it is removed from the model. Another name for this method is Least Absolute Shrinkage and Selection Operator, or LASSO for short, which refers to the method that selects certain features and removes others [21]. ℓ_1-Regularization is based on the linear regression model, where one assumes a linear relationship between the feature variables x_p and the target y, with a certain coefficient β_p and an error term ϵ. With a regularization hyperparameter λ, we can define the ℓ_1 penalty term as

$$\ell_1 = \lambda(|\beta_1| + |\beta_2| + ... + |\beta_p|). \tag{3.7}$$

The resulting objective function that we seek to minimize for an ℓ_1-Regularization is the sum of the ℓ_1-term and the residual sum squares(RSS) [38],

$$\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}. \tag{3.8}$$

Due to the absolute value terms, the objective function is not differentiable at zero, and while there are methods for solving this, such as Karush-Kuhn-Tucker (KKT) theory, a closed-form solution does not exist. Another drawback is that when there exists many features than there are samples, i.e. $p > n$, the ℓ_1-regularization tends to behave poorly, selecting to few or to many features. The strength of this method, is the creation of a sparse formulation, which may significantly decrease the computational demand. The ℓ_1-regularization model is still linear, but the ℓ_1 penalty forces non-contributing terms to zero, simplifying and in many cases improving the model. Finding an optimal balance between high regularization, which removes more features, and not simplifying the model too much is crucial; hence, the tuning of λ is vital for a satisfactory result. A higher value of λ imposes a stronger penalty, and more coefficients will be shrunk to zero, while a smaller value results in the ℓ_1 term behaving more like ordinary least squares and the model approaches a standard linear regression.

3.2.3 Ridge regression / ℓ_2-Regularization

A task containing many features makes the model more complicated and demands higher computing power, however, in the cases where all these features are relevant, instead of removing them we can instead choose to decrease their impact. This is known as a ℓ_2-regularization. Due to its origins in ridge analysis, a statistical method for handling multicollinearity, ℓ_2 regularization is also known as ridge regression. Similarly to ℓ_1, we are still looking at a linear regression with features x_p and targets y. The penalty term ℓ_2 consists of the residual sum of squares and the squares of the coefficients [21],

$$\ell_2 = \lambda(\beta_1^2 + \beta_2^2 + ... + \beta_p^2). \tag{3.9}$$

This will heavily penalize the higher terms, but has little effect on smaller terms; this means no coefficients are shrunk to zero. The hyperparameter λ regulates the model and controls the severity of the penalty terms and regularization. The objective function for an ℓ_2-regularization is,

$$\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}. \tag{3.10}$$

The addition of a squared term in the objective function means the derivative of the objective function becomes $-2\mathbf{X^T}(\mathbf{y} - \hat{\mathbf{y}}) + 2\lambda\beta$ with the closed form solution, $\hat{\beta} =$

$(\mathbf{X^T X} + \lambda \mathbf{I})\mathbf{X^T y}$ [39]. The Ridge regression solves the case of $\mathbf{X^T X}$ approaching a singular matrix for high multicollinearity by creating the new matrix $(\mathbf{X^T X} + \lambda \mathbf{I})$ with $\lambda \geq 0$. It can be shown that in a certain range of values, $0 < \lambda < \lambda^*$, the resulting mean square error for ridge regression is smaller than the corresponding least square error [40].

ℓ_2-regularization aims to make the model more accurate while still keeping all features. While ℓ_1-regularization tends to solve multicollinearity by arbitrarily keeping only one of the collinear features, ℓ_2-regularization shrinks all instead; this is the major difference between these techniques. ℓ_2-regularization tends to outperform ℓ_1-regularization in the cases of many correlated features due to its stable assignment of weights to all features. Compared to its LASSO counterpart, ℓ_2-regularization does not produce sparsity.

3.2.4 Elastic Net

For the case of selecting a subset of features, LASSO performs well, while ridge regression is used for keeping all features in the model. Comparisons between ℓ_1 and ℓ_2 regularizations have shown that neither of these methods can conclusively be said to outperform the other; it is all dependent on the task [41] [42]. A method which combines the strengths of LASSO and ridge regression is the so-called elastic net. The elastic net combines the penalty terms of the ℓ_1 and ℓ_2 regularization. The ℓ_1 term allows it to shrink some coefficients to zero, while the ℓ_2 term means the elastic net can group correlated features and shrink them together, keeping them in the model. This solves LASSO's drawback for the case of $p > n$ and ridge regressions inability to select a subset of features. For a dataset with a large amount of features, where some are correlated, the elastic net can provide a powerful tool for improving the ML model. The loss function for an elastic net is tuned by hyperparameters λ and α [38],

$$\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda(\alpha \sum_{j=1}^{p} |\beta_j| + (1-\alpha) \sum_{j=1}^{p} \beta_j^2) \right\}. \tag{3.11}$$

The regularization strength is determined by λ while α is known as the mixing parameter, or ℓ_1 ratio and determines if the elastic net should behave more like ℓ_1 or ℓ_2-regularization. In the cases where we set $\alpha = 0$, the elastic net mimics an ℓ_2-regularization, for the case of $\alpha = 1$, it results in an ℓ_1-regularization. In practice, due to its stability and combination of both models' strengths, the elastic net is often preferred over both LASSO and ridge regression, especially in high-dimensional settings with potentially correlated predictors.

3.3 Selected Ensemble and Tree-Based Models

In cases where the linear methods perform poorly, it is necessary to consider models that can identify and capture non-linear relationships in the data. Ensemble and tree-based methods are widely used for this purpose due to their strong performance and robustness in their predictions. Ensemble methods combine the predictions of multiple base models to produce a single output. Methods which rely on decision trees as their fundamental predictive units are called *tree-based* models. A decision tree is a supervised learning model which recursively partitions the feature space into smaller and more homogeneous regions by applying a sequence of simple decision rules. The resulting structure resembles a tree, where internal nodes represent decision rules, branches correspond to outcomes of these rules, and terminal nodes (referred to as leaves) produce the final prediction.

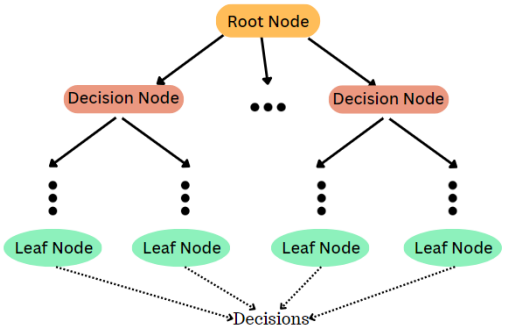


Figure 3.3: A Decision Tree

Decision trees are intuitive and easy to interpret, but when used individually they tend to have high variance and are prone to overfitting the training data [21, 43] Using an ensemble of decision trees has proven to be very useful in machine learning, and we will present two separate methods in this section: Random Forest and XGBoost. While both models utilize an ensemble of decision trees, the construction and generation of a prediction are fundamentally different.

3.3.1 Random Forest

Random Forest (RF) is an ensemble learning method that constructs a large number of decision trees and combines their predictions through averaging (for regression). The core idea behind RF is to introduce randomness during training in order to decorrelate individual trees, thereby reducing the variance of the ensemble without significantly increasing bias [44]. Each tree is fit on a bootstrap sample (sampling rows with replacement), and at every split the tree considers only a

random subset of features (the "feature bagging" step). These two sources of randomness decorrelate the trees: individual trees have high variance, but averaging their predictions reduces variance without greatly increasing bias. The prediction of a Random Forest regressor is obtained by averaging the predictions of all individual trees. While each tree on its own may be a weak and noisy predictor, their aggregation yields a stable and accurate model. Important hyperparameters include the number of trees $n_{\text{estimators}}$, the maximum tree depth, the minimum number of samples per leaf, and the number of features considered at each split m_{try}. Increasing the number of trees generally improves stability at the cost of increased computational time.

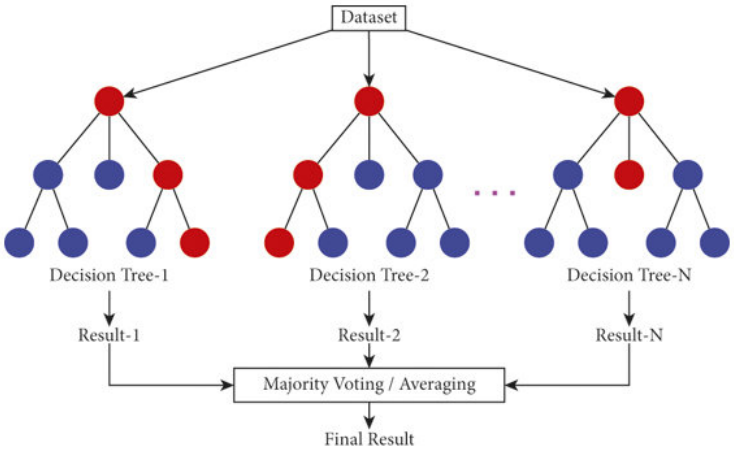


Figure 3.4: Illustration of a Random Forest Model

Because decision trees partition the feature space using threshold-based decisions on individual features, Random Forests can naturally model non-linear relationships and high-order interactions between variables. They require no feature scaling and are invariant to monotonic transformations of the inputs. Random Forests also provide estimates of feature importance, which can be useful for model interpretation and diagnostics, although such measures may be biased in the presence of strongly correlated features [21]. Unlike linear models, RFs do not extrapolate outside the support of the training data; predictions are averages of seen regions. In sequence problems like per-lot quality prediction from sensor streams, RFs operate on fixed-length tabular features, so we summarize each lot's time series into statistics (mean, std, extremes, ramp-rate, quantiles). This transforms variable-length sequences into comparable vectors and avoids leakage by computing statistics only within each lot's time window. To evaluate fairly, the split must be by lot (or time) rather than by row so that all timestamps from a

given lot stay on the same side of the split.

3.3.2 XGBoost

An alternative tree-based model is the gradient boosting method. Instead of continuously building independent random trees like the random forest model, gradient boosting instead builds one tree at a time, with each tree aiming to improve upon the last. The first tree, known as the base learner makes a simple prediction, for example the mean value of the target variables in the training data. The second tree, looks at the residual between the base learner's prediction and the actual value, which is used as the new template that the next tree strives to improve upon. The successive trees created to improve the previous setups result are known as weak learners [45]. The act of minimizing the error of the prediction and the ground truth value, i.e., minimizing the loss function is done with the gradient descent algorithm.

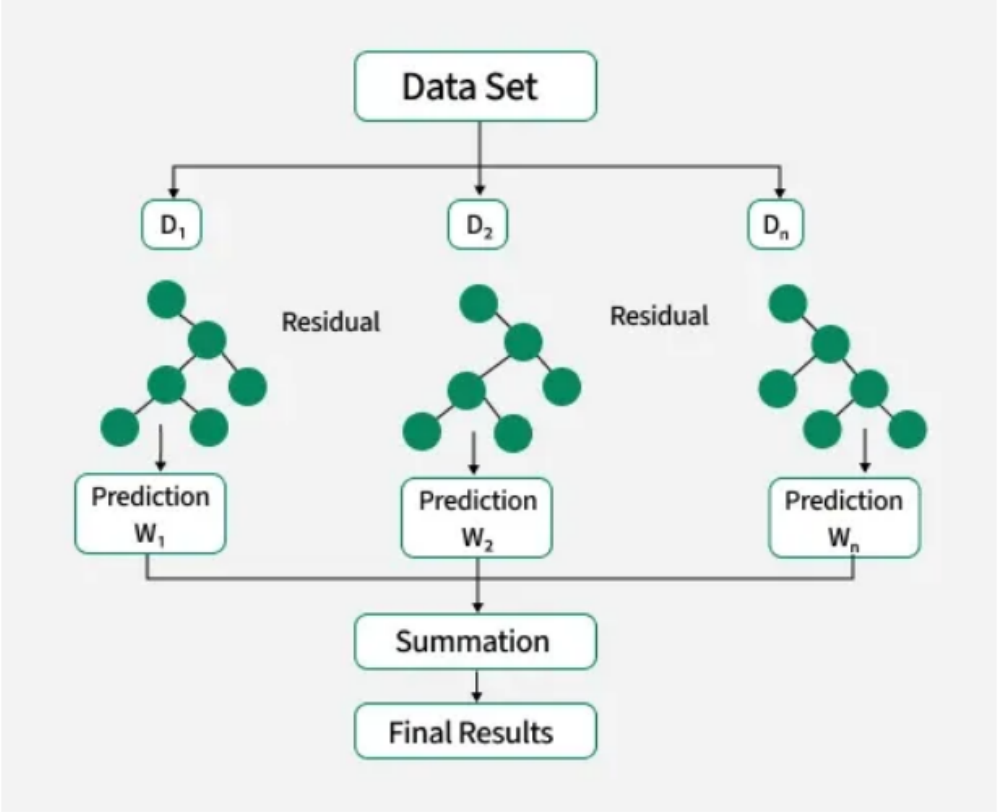


Figure 3.5: Chart of XGBoost's decision trees[46].

As the model builds into more trees, the predictions will improve, and the

combination of weak learners create what is known as a strong learner. The final prediction is the sum of contributions from all trees,

$$\hat{y}_i = \sum_{n=1}^{N} f_n(x_i), \tag{3.12}$$

where each function f_m represents a weak learner and N is the total number of trees. Due to the model creating newer trees sequentially based on the previous error instead of them all simultaneously, gradient boosting is generally more demanding than random forest. An improvement to the gradient boosting method was created in the 21st century in the form of XGBoost(eXtreme Gradient Boosting). XGBoost is an open-source library that takes the base algorithm of gradient boosting and implements features such as built-in ℓ_1 and ℓ_2 regularization, to avoid overfitting and the ability to handle missing values in the data. Parallel processing and more efficient computations makes XGBoost optimal for handling large numbers of data [47].

The amount of hyperparameters for the XGBoost model is large, a brief explanation of their impacts are given in the following table.

Hyperparameter	**Description**
Number of estimators	Regulates the total number of boosting trees used in the ensemble.
Learning rate	Determines how much each individual tree contributes to the final model.
Max depth	Sets the maximum depth for each decision tree. This regulates model complexity and helps to prevent overfitting.
Subsample	Defines the number of samples randomly selected to train each tree. Low subsamples improve generalization, but can cause underfitting.
Colsample bytree	Determines the fraction of features randomly sampled when constructing each tree, reducing correlation between trees and improving robustness.
Regularization Alpha	Controls the strength of ℓ_1 regularization.
Regularization Lambda	Controls the strength of ℓ_2 regularization.

Table 3.1: Overview of XGBoost hyperparameters with short summaries.

3.4 Selected Sequence Models

Recurrent neural networks model sequences by updating a hidden state $\mathbf{h}_t = \phi(\mathbf{W}_x\mathbf{x}_t + \mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{b})$ and producing outputs $\mathbf{y}_t = g(\mathbf{W}_y\mathbf{h}_t + \mathbf{c})$, where ϕ and g are nonlinearities. Training uses backpropagation through time (see sec. 3.1.4). In practice, *vanishing* and *exploding* gradients make long-range credit assignment difficult, which motivated gated architectures that regulate information flow through time. Gradient clipping and careful initialization partly mitigate these issues but do not fully solve them for plain RNNs. [48, 49, 50].

3.4.1 Gated Recurrent Unit

Recurrent neural networks (RNN) model sequences by maintaining a hidden state that is updated step-by-step as new observations arrive. Vanilla RNNs struggle with long-range dependencies because gradients either explode or vanish during back-propagation through time. Gated Recurrent Units (GRUs) mitigate this with gates that regulate information flow, letting the model retain or discard past information as needed. A GRU cell has two gates: the update gate z_t, and the reset gate r_t. Given an input vector x_t and the previous hidden state h_{t-1}, the reset gate controls how much of the old state to use when computing a candidate state $\tilde{h}_t$, while the update gate interpolates between the old and candidate states to form the new hidden state h_t. This can be written with the following expressions:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\,\mathbf{x}_t + \mathbf{U}_z\,\mathbf{h}_{t-1} + \mathbf{b}_z)\,, \tag{3.13}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\,\mathbf{x}_t + \mathbf{U}_r\,\mathbf{h}_{t-1} + \mathbf{b}_r)\,, \tag{3.14}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h\,\mathbf{x}_t + \mathbf{U}_h\,(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h)\,, \tag{3.15}$$

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t. \tag{3.16}$$

where $\sigma(\cdot)$ is the logistic sigmoid function and $\odot$ denotes element-wise multiplication. The parameters $\mathbf{W}_{(\cdot)} \in \mathbb{R}^{H\times F}$ and $\mathbf{U}_{(\cdot)} \in \mathbb{R}^{H\times H}$ are *learnable weight matrices* (input-to-hidden and hidden-to-hidden, respectively); $\mathbf{b}_{(\cdot)} \in \mathbb{R}^H$ are biases. All these parameters (and the biases) are learned during training via backpropagation through time. Intuitively, r_t "forgets" irrelevant history when forming the candidate, and z_t chooses how much to overwrite the state.

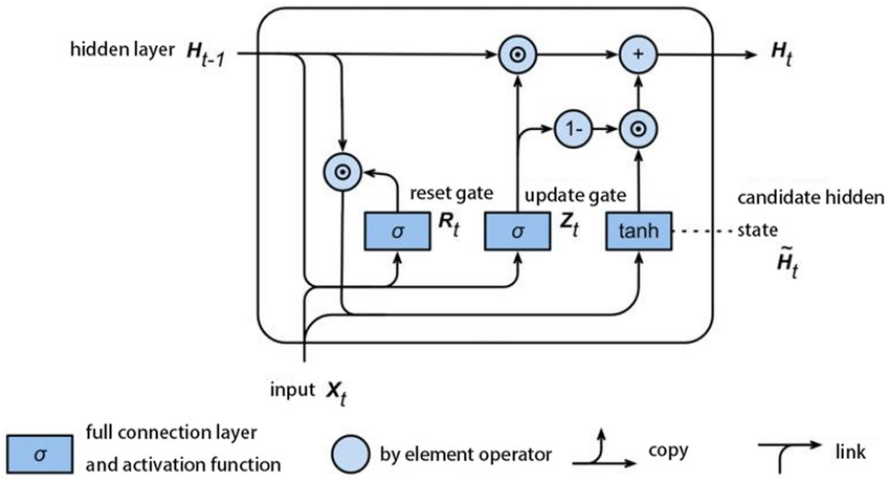


Figure 3.6: Illustration of the GRU model

Compared to LSTMs, GRUs are slightly simpler (fewer gates, no separate cell state), which reduces parameters and can train faster while retaining the ability to model medium/long temporal dependencies. [51, 14] Our data are multi-sensor time series with relatively few lots per partition (see chapter 4); GRU's lower parameter count reduces overfitting risk and training time while still capturing furnace/thermal dynamics. As mentioned previously in sec. 2.2, prior work reports successful GRU applications to industrial furnaces and thermal processes, supporting our architectural choice. [13]

3.4.2 Long-Short Term Memory

A more complex model than GRU is the Long-Short Term Memory method. While GRUs take into account historical data through its hidden state, it may struggle for more complex large datasets and is limited by its two states. The augmented version, Long-Short Term Memory or LSTM implements three gates: forget, input and output, as well as an explicit state for long-term memory. As the model updates, the gates regulate which data to keep in the long-term memory state. The result of this added state means our model can continuously process new data and save the necessary features in the memory state. Adding a separate cell to handle memory leads to LSTM models performing with higher accuracy for larger problems with long-term time dependencies. The addition of a third cell, compared to GRUs two, does however, come with a computational cost. An LSTM is only suited for machine learning problems with complicated, long-term time dependencies, in other simpler cases, the GRU model will give similar results within shorter time and is thus to be preferred [50, 14].

Chapter 4

Data

This chapter describes the datasets used in this study, their scope and measurement conventions, and the variables modeled as outcomes. We report sources, coverage, and basic statistics, while all preprocessing and modeling procedures are deferred to Section 5.

4.1 Data Sources and Scope

To ensure traceability throughout the production process, Höganäs divides the powder output into discrete batches of approximately 30 tonnes. Each batch is referred to as a *production lot* (or simply *lot*) and is assigned a unique identifier, denoted as `LotID`. This identifier enables consistent tracking of material flow and facilitates the linkage between process data and laboratory analyses.

Production (Process) Data

The reduction–annealing work centres `SPDFUR22` and `SPDFUR24` log multivariate time series of furnace and utility signals. A total of 131 tags are sampled with a sampling frequency of five minutes, in the time interval from January first 2024 through the data–collection date (2025-10-08). Tags are organized by physical section: *CoolZone 1–4*, *HeatZone 1–8* (separate top/bottom measurements per zone), *General*, *Media*, *Input*, and *Output*. Many control loops expose three channels per signal—set-point (SP), process value (PV), and controller output (Out). In this study we represent the realized operating conditions by the PV channels only. In the cooling sections, internal temperatures are not measured directly; proxies such as cooling-water temperature and flow are logged.

Laboratory (Quality) Data.

Powder samples taken from production lots are analysed in the laboratory. Seventeen properties are available in the source system; this thesis focuses on six outcome variables relevant for press–sinter performance (Section 4.3).

4.2 Tag Taxonomy and Measurement Channels

Table 4.1 lists the tag groups present on both lines. After restricting to PV channels and excluding purely administrative identifiers at analysis time, the analysed signal set comprises a total of 61 PV tags spanning thermal (HeatZone Top/Bot), cooling proxies (CoolZone water temperature/flow, dew point), and plant utilities (Media, e.g., process water, currents, gas flows). Exact tag names follow the `<WorkCentre> > <Section> > <Signal>-PV` convention used by Höganäs AB (e.g., `SPDFUR24>HeatZone07>TempBot-PV`).

Group (section)	#PV tags	Examples
HeatZone 1–8 (Top/Bot)	*[16]*	`TempTop-PV`, `TempBot-PV`
CoolZone 1–4	*[16]*	`WaterTemp-PV`, `WaterFlow-PV`, `Dewpoint-PV`
Media (utilities)	*[16]*	`ProcessWater(8m)-PV`, `OxygenLevelCoolZone-PV`
General	*[9]*	`BeltSpeed-PV`, `VolumeWeight-PV`
Input / Output	*[4]*	`BinWeight-PV`, `OutletWeight-PV`
Total (PV used)	**61**	

Table 4.1: Tag groups available from the furnace control system. Counts refer to PV channels analyzed in this work.

4.3 Target Variables (Laboratory Outcomes)

During the reduction annealing process, samples of the powder are being collected throughout the process to analyse a number of properties. In total, seventeen numerical values are measured, but in consolidation with Höganäs, six of the more important measurements were used as targets for the modelling:

- **Particle size classes (wt.%):** -45 μm *(SH-45)*, $+75-106$ μm *(SH7510)*, $150-212$ μm *(SH1521)*.
- **Chemistry (wt.%):** Carbon *(CHC)*, total oxygen *(CHO)*.

- **Green density:** g/cm^3 at 600 MPa compaction *(PHGD6)*.

All targets are reported by the lab against the lot identifier. Main results are presented in original units for process interpretability; normalized-space metrics appear only for diagnostic comparison in later sections. The target variables are measured using different test, such as sieve test for particle sizes or chemical analysis for the powder compositions, and during the $\sim$ 20 hour run time of a production lot, a number of samples are collected on which different analyses are performed. The time duration it takes to collect a sample from the production line is around 4 hours. The iteration of lot sample is logged by using the tag `lopnr`. Since the tests performed on the sample differ in complexity and time consumption, not all the tests are performed every time. One additional demarcation is thus to only use values from the analyses done on the first of each lot sample, since we have been informed that the first sample of each lot is always tested for all seventeen properties.

Green Density

Green density (GD) refers to the bulk density of a powder compact immediately after uniaxial pressing (see fig. 4.1). The GD aims to reflect how efficiently the powder particles pack and deform under applied pressure. It is measured as $\rho_g = m/V_b$, where m is the compact mass and V_b is the bulk volume, using a compact pressed at a specified pressure. In our case, this pressure was 600 MPa. The GD is often reported relative to the theoretical density ρ_{th} to reflect the powders compressibility, which is its ability to densify under pressure. A higher GD, and thus compressibility, are generally desired for sintered products since more inter-particle contact provides stronger and more consistent sintered components, better resistant to damages. For water-atomized low-alloy powders, such as the Astaloy powders studied in this report, the surface oxides and work-hardening from atomization and drying can depress the compressibility. Therefore *reduction annealing (see sec. 2.1.1)* is used to clean the surface from the oxides, restoring ductility, and raising ρ_g for future sintering. [5, 2, 3]

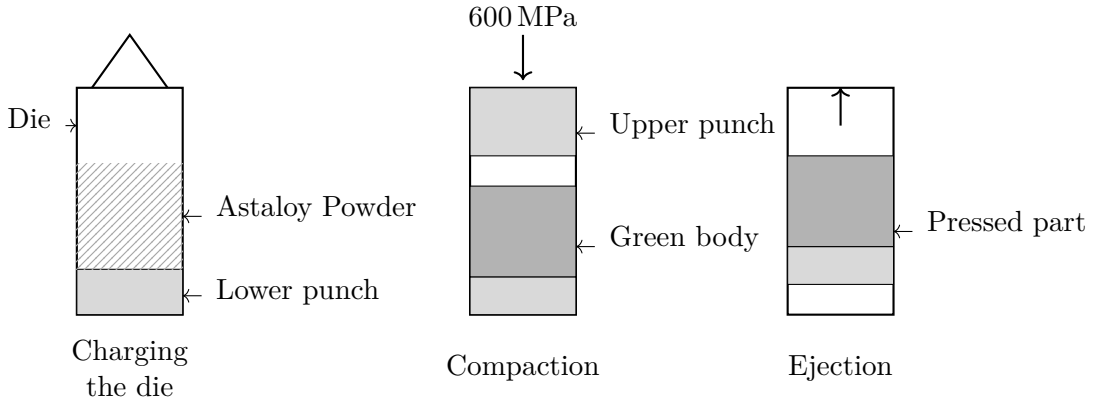


Figure 4.1: Schematic of uniaxial pressing of Astaloy powder: charging the die, compaction at 600 MPa, and ejection of the green compact.

Particle Size Distribution

Three out of our six target variables are regarding the Particle Size Distribution (PSD) of the powder. The PSD is reported as weights fraction from standard sieve analysis of lot samples. We use three informative classes **SH-45** ($< 45.000\,\mu m$), **SH7510** ($+75.000\,\mu m$ to $106.000\,\mu m$), and **SH1521** ($150.000\,\mu m$ to $212.000\,\mu m$). If the powder is too fine, (high SH-45 weight percentage) this will raise the specific surface area and absorbed oxygen (surface oxides) thus reducing the flowability. If the particles are too large on the other hand (high SH1521 weight percentage), they pack less efficiently, increasing the bulk volume V_b and leading to worse compressibility [5].

4.4 Data Challenges

There are a number of known limitations from the challenges in the data, in this section we will go over a few of those.

Number of Samples

While the sampling frequency of one minutes is more than adequate for a process that takes around 24 hours, the collection of production data only goes back to the start of 2025. This limits the amount of matching input-output data pairs that are available to train the model.

Linkage fidelity

Accurate lot-level alignment requires robust joins between *Input*/*Output* identifiers and lab records. Inconsistent naming, missing lot markers, partial runs, or rework can produce ambiguous intervals, which would skew the results. Ambiguities are resolved by rule-based checks and unresolved lots are excluded.

Proxy Measurements in Cooling

In the cooling sections of the furnace, internal material temperatures are not measured directly. Instead, indirect proxy signals such as cooling-water temperature and water flow are recorded. These variables reflect the heat removal rate but only indirectly represent the actual thermal state of the powder, which introduces additional uncertainty and noise into the measurements.

Missingness and Outliers

Faulty measurements, start-up/shutdown states, and maintenance events introduce gaps and outliers. Data with missing values is filtered out, while outliers are identified early on in preprocessing.

Data Scale Magnitudes

Due to the target variables all being vastly different, the scale of their sizes are not identical. This can confuse the machine learning models and makes visualizing its performance more challenging to compare its variables. This is solved by normalizing all data before it is sent into the model for training; this makes the inputs and outputs scale invariant.

4.5 Data Visualization

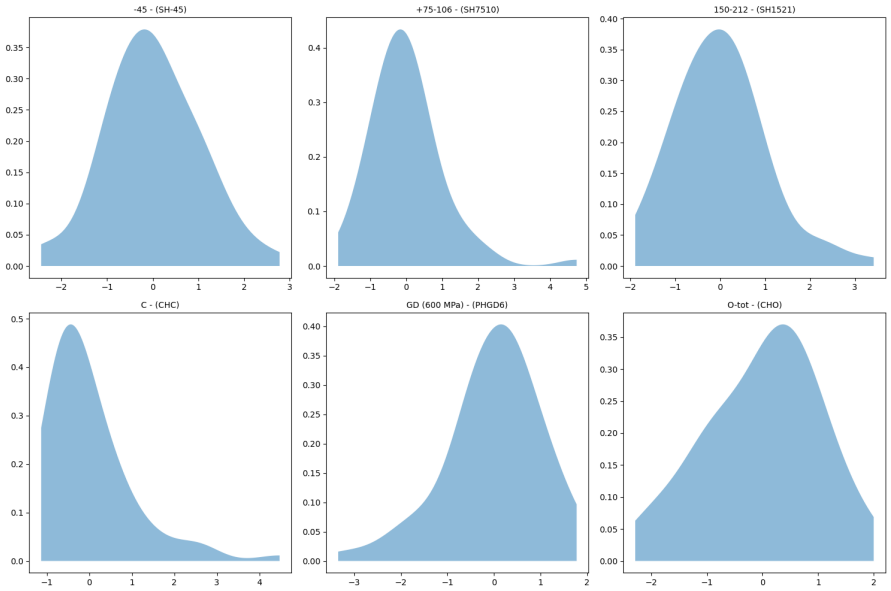
A short visualization of the data is presented in this section, to give an idea of the structure of the target variables in the laboratory data and the 61 features extracted from the production data. For the 6 target variables, which consists of size distributions, green density, and chemical composition properties, all outputs will show unique scales and distributions. To gain insight into how the data is distributed for each respective variable, the normalized std is used,

$$\text{Normalized Std} = \frac{\sigma}{\max(y) - \min(y)}. \tag{4.1}$$

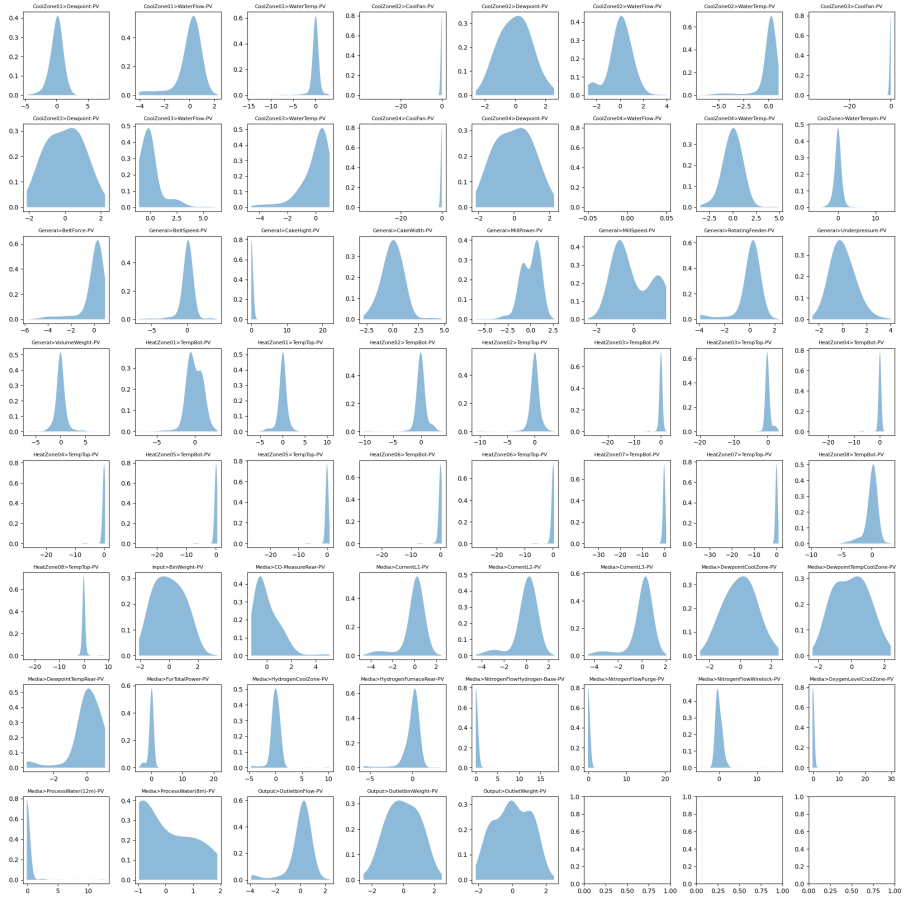
Output	**Mean**	**Std (σ)**	**Normalized Std**
Output 1: SH-45	14.351	1.895	0.126
Output 2: SH7510	28.552	2.290	0.182
Output 3: SH1521	8.410	1.640	0.143
Output 4: PHGD6	6.955	0.022	0.201
Output 5: CHC	0.003	0.002	0.126
Output 6: CHO	0.152	0.033	0.192

Table 4.2: Mean, std, and normalized std for each output.

This table highlights the importance of normalization, for outputs 5 and 6, which are of considerably smaller scale than the other outputs; both the mean and the std σ are many magnitudes smaller. The normalized std shows the true distribution for all variables and reveals that output 4 has the highest variance while outputs 1 and 5 are the least varying. A visualization is presented for both the 6 target variables as well as the 61 features after normalization using density histogram plots.



(a) Target variables density distribution after normalization.



(b) Production features density distribution after normalization, 61 variables total leaves 3 empty boxes in an 8x8 grid.

Chapter 5

Methodology

This chapter aims to describe the methodology and demonstrate the approaches taken during this thesis.

5.1 Data Preprocessing

The preprocessing pipeline cleans and converts raw furnace time series and laboratory results into analysis-ready datasets for two separate instances, *per-lot* and *per-timestamp*. The per-lot dataset consists of features for classical regressors and tree baselines, such as LR, Ridge/LASSO/Elastic net, RF and XGBoost. The per-timestamp (or sequence) dataset is modelled to fit the sequence models GRU and LSTM. The implementation of the preprocessing is designed to be reproducible and to guard against information leakage: all normalizers and dimensionality-reduction transforms are fit on the training lots only, and the exact feature order and scalers are persisted for inference reuse. The pipelines follow the same workflow as seen in figure 5.1.

Analysis Data

The analysis data, or targets, are obtained through work centre export in the form of a `.csv` file. The first step is to filter the data to dispose of irrelevant information which is not regarding any of the $Product \times WorkCentre$ combinations. After this, the data is filtered to only include the first sample for each lot so that no target variable has a missing value due to the omission of analyses, as discussed in section 4.3. To ensure reliable joins between production and laboratory data, all lot identifiers were standardized into a single canonical key, denoted LotK:

$$\text{LotK} = \textsc{Upper}(\textsc{Trim}(\textsc{String}(\textsc{Round}(\textsc{ToNumeric}(\mathsf{LotID}))))) .$$

This normalization removes formatting inconsistencies by converting lot identifiers to a consistent string representation (e.g., handling numeric formats, rounding, trimming whitespace, and enforcing upper-case text). Non-numeric, empty, or missing lot identifiers are coerced to missing values during normalization and are therefore excluded from subsequent joins. The resulting key allows robust matching across data sources even when original identifiers differ in format. The laboratory targets are then also coerced to numeric and filtered to the six outcomes used in this thesis: -45 (SH-45), $+75-106$ (SH7510), $150-212$ (SH1521), carbon (CHC), green density at 600 MPa (PHGD6), and total oxygen (CHO).

Production Data

The production historian extract is obtained from two `.csv` files, one for each of the work centres. From the production file we keep all `-PV` (process-value) columns detected by suffix as well as the lot tag `SPDFUR24>Output>LotNbr` for the lot identifier and the time index `StartTimeUTC` are also retained. This yields a *per-timestamp* view (one row per timestamp with all PVs and the lot), it is the exact input matrix for sequence models (the code also persists the PV channel order in so that training and inference use identical feature ordering, which the sequence models require).
Per-lot aggregation for tabular models For classical regressors we compress each lot's PV time series to a single feature vector via descriptive statistics. After coercing PVs to numeric, we group by LotK and compute, for every PV channel x_t in the lot interval,

$$\mu = \mathrm{mean}(x_t), \quad \sigma = \mathrm{std}(x_t), \quad x_{\min} = \min_t x_t, \quad x_{\max} = \max_t x_t, \quad \tilde{x} = \mathrm{median}(x_t),$$

and a simple dynamics proxy, the *mean absolute ramp-rate*:

$$\mathrm{marr}(x) \;=\; \frac{1}{T-1}\sum_{t=2}^{T}\Big|x_t - x_{t-1}\Big|.$$

The per-lot features are then inner-joined with the targets on the canonical lot key, yielding the feature matrix $\mathcal{X} \in \mathbb{R}^{H\times 367}$ (61 PV tags $\times$ 6 descriptive statistics + `LotK`), where H is the number of overlapping lots between production- and analysis data. The inner join avoids label leakage from imputation and guarantees each training example has both inputs and outputs. We also construct fixed splits for the lots used during training and testing. The same split is reused across all models (Random Forest, Elastic Net, GRU/LSTM) to ensure that a lot never appears in both train and test, and that the sequence models never see future time steps from a test lot during training. All preprocessing transformers (standardization, target normalization, PCA) are fit on the training lots only and applied to test.

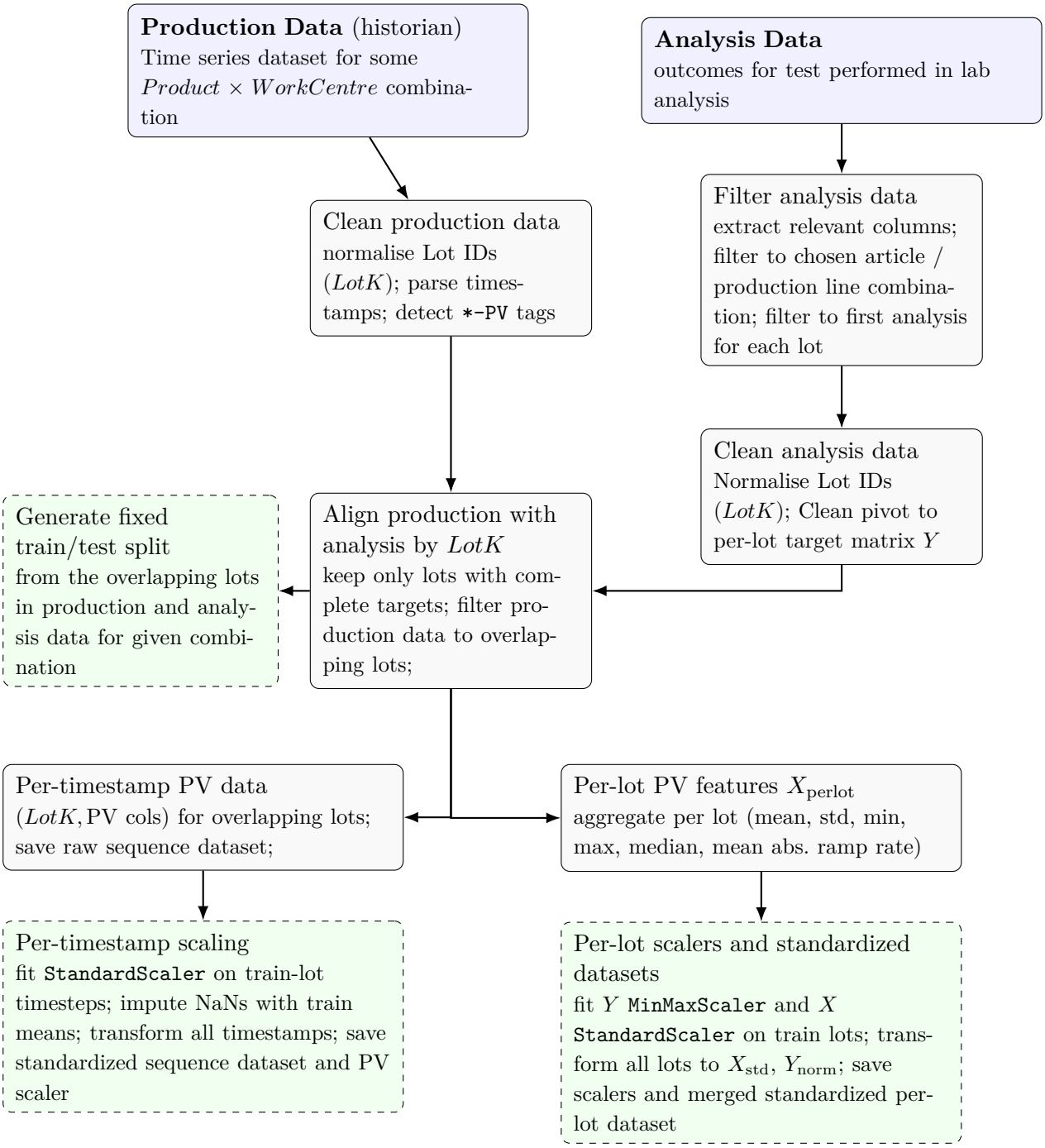


Figure 5.1: Preprocessing workflow for each furnace–product combination. Production historian data are cleaned and aligned by lot with cleaned lab analysis targets, then split by lot into fixed train and test sets. The left branch produces per-timestamp sequences; the right branch produces per-lot tabular features, both branches produce standardized dataset with scalers.

5.1.1 Handling of Missing Values and Outliers

Missing values and outliers were handled as part of the preprocessing pipeline prior to model training. Missing sensor readings primarily arose from temporary communication losses or start up/shut down of the annealing furnace. The missing values arising from temporary communication loss where short gaps in the time-series data were handled implicitly through aggregation at the lot level whereas the start up/shut down of the furnace or changing productions lots were identified by "faulty" values in the lot identifier (such as zero, fractions or `NaN`) and where thus handled by the exclusion rows as mentioned in 5.1.

5.1.2 Data Splitting

To prevent information leakage, data splitting is performed *by lot* rather than by individual timestamps. This ensures that all observations originating from the same production lot are assigned exclusively to either the training or the test set. Splitting at the timestamp level would otherwise allow the model to observe parts of a lot during training and be evaluated on other parts of the same lot. We construct a fixed split file (`lot_splits.json`) containing training and test lot identifiers; the same split is reused across all models. This ensures that (i) a lot never appears in both train and test, and (ii) sequence models never see future timesteps from a test lot during training. All preprocessing transformers (standardization, target normalization, optional PCA) are fit on the training lots only and applied to test. We report the number of lots and timestamps per split, and verify there is no temporal leakage by sorting within-lot by time during sequence building.

5.1.3 Feature Engineering

We work with two representations of the same furnace signals. (1) **Per-lot tabular features** (for tree/linear models): for each lot we aggregate numeric PV tags over the lot window using mean, standard deviation, min, max, median, and a simple ramp-rate proxy (mean($|\Delta \text{PV}|$)) to capture dynamics. These features are then standardized (train only). Targets are normalized to $[0, 1]$ (train only) to stabilize multi-output regression. (2) **Per-timestamp sequences** (for GRU/LSTM): we retain the full time series per lot. Each timestamp vector is standardized using a per-timestep `StandardScaler` fit on training timesteps. Optionally, we apply PCA to reduce input size and denoise before feeding sequences to the RNN (see Sec 5.1.4). All feature construction is deterministic and reproducible; we persist the exact PV column order (`pv_feature_names.json`) so downstream models see a consistent layout.

5.1.4 Dimensionality Reduction

Principal Component Analysis (PCA) is used to reduce feature dimensionality while retaining most of the variance in the data. Given a zero-mean, standardized feature matrix $X \in \mathbb{R}^{n \times d}$, PCA finds an orthonormal projection $W_k \in \mathbb{R}^{d \times k}$ that maximizes the projected variance,

$$W_k^\star \;=\; \arg \max_{W^\top W = I_k} \; \mathrm{Var}(XW).$$

Each principal component captures a portion of the total variability in the original data. The *explained variance* tells us how much "signal" is kept by the components we retain. For example, retaining components that together explain about 95% of the variance means we keep most of the useful variation while discarding mostly redundancy and noise.

How we apply PCA in this project. We work with two representations of the same process data:

1. **Per-lot (tabular) features**: For each lot, we compute aggregate statistics of PV tags over the lot window (mean, standard deviation, min, max, median, and a simple ramp-rate feature). After standardizing these features, we optionally apply PCA to reduce dimensionality before feeding them to non-sequence models (e.g., Random Forest, Elastic Net). Trees do not strictly require PCA, but reducing redundancy can still help with speed and stability.

2. **Per-timestamp PV features**: For sequence models (GRU/LSTM), we keep the time series per lot. At each timestamp we first standardize the PV tags using a scaler fit on training data, and we optionally apply PCA to project each timestamp to fewer components. This lowers the input size to the RNN and can act as a mild denoising step.

To avoid information leakage, all scalers and PCA transforms are fit *only on the training split* and then applied to the test split.

Model-facing choices. The results of the PCA analysis are reported in Section 6.5. Note however that the PCA was only performed during preprocessing (as seen in fig. 5.1) and the ML models evaluated during this thesis (Section 6.1) do not utilize the dimensionality reduction. The motivation for this decision is that the computational time of the model training was not as excruciatingly long as expected and thus did not warrant the need for dimensionality reduction, though it may have been beneficial in reducing noise.

5.2 Model Development

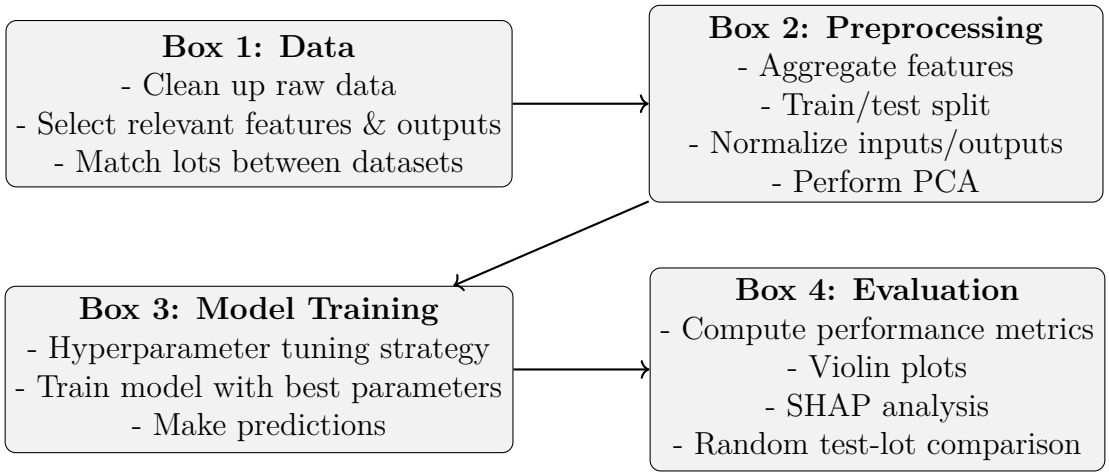


Figure 5.2: Machine Learning Pipeline Overview

5.2.1 Implementation Tools

The machine learning models were implemented in python using well-known and established machine learning libraries. Scikit learn, is an open source and easy-to-use library that covers most classical machine learning algorithms with built-in cross validation and hyperparameter tuning. It also includes tools for creating test/train splits, evaluation metrics, and normalization scalers. For the cases where a model was not available on scikit-learn, PyTorch was the option chosen. Famous for its ability to handle large amounts of data and computationally heavy models, making it ideal for deep learning and neural networks.

Implemented Models

The following models were implemented using Scikit Learn: Multiple Linear Regression, ℓ_1-Regularization, ℓ_2-Regularization, Elastic Net, and Random Forest. For the potentially more demanding recurrent neural network models, GRU and LSTM, PyTorch was used. Finally, XGBoost has its own library, created by its developers which is python compatible.

Chapter 6

Experiments

While data exists for two furnaces SPDFUR22 and SPDFUR24 as well as for two seperate products Asataloy CrA and Astaloy CrM, due to the high number of tables and figures needed to present the results, all following experiments, results, discussions and conclusion will be shown for exclusively the Astaloy CrM product for the furnace SPDFUR24. This is the largest of the datasets and is thus the most representative for evaluating our goals.

6.1 Performance of Final Models

Output (Target)	Linear Reg.	LASSO	Ridge	Elastic Net	LSTM	XGBoost	Random Forest	GRU
SH-45	1.353	**0.897**	2.721	2.109	2.148	2.29	1.838	2.5474
SH7510	**0.679**	0.757	1.130	1.114	1.401	1.237	1.609	1.6845
SH1521	1.512	**0.858**	1.281	1.312	1.339	1.131	1.587	1.3911
PHGD6	2.154	1.675	0.019	0.016	**0.014**	**0.014**	0.018	0.021
CHC	2.152	1.873	**0.002**	**0.002**	**0.002**	**0.002**	0.0025	0.003
CHO	1.362	1.044	0.033	**0.026**	0.032	**0.026**	0.032	0.0313
Mean RMSE	1.58	1.18	0.86	**0.76**	0.82	0.78	0.85	0.95

Table 6.1: Root Mean Square Error (RMSE) . Lower is better. Best per row in **bold**.

Output (Target)	Linear Reg.	LASSO	Ridge	Elastic Net	LSTM	XGBoost	Random Forest	GRU
SH-45	1.086	**0.717**	2.020	1.696	1.771	1.786	1.583	1.7967
SH7510	0.554	**0.510**	0.976	0.965	1.015	0.885	1.19	1.059
SH1521	1.077	**0.554**	1.109	0.853	1.061	0.777	0.8081	1.0014
PHGD6	1.540	1.304	0.015	0.012	0.011	0.011	0.015	**0.0099**
CHC	1.953	1.376	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**	0.0035
CHO	1.119	0.818	0.025	**0.021**	0.026	0.022	0.026	0.0273
Mean MAE	1.22	0.88	0.69	0.59	0.65	**0.58**	0.60	0.65

Table 6.2: Mean Absolute Error (MAE). Lower is better. Best per row in **bold**.

Output (Target)	Linear Reg.	LASSO	Ridge	Elastic Net	LSTM	XGBoost	Random Forest	GRU
SH-45	34.075	109.56	14.073	12.632	13.191	13.232	**11.30**	12.2435
SH7510	25.877	88.402	3.48	3.436	3.53	**3.1**	4.28	5.3739
SH1521	70.844	90.96	13.826	10.696	13.237	16.15	**10.4865**	12.9339
PHGD6	1.366	152.587	0.218	0.171	0.161	0.156	0.21	**0.1417**
CHC	120.26	120.159	58.048	48.599	47.883	**35.838**	41.19	86.2319
CHO	55.268	135.629	17.246	**14.907**	18.146	15.36	19.07	18.2919
Mean SMAPE	42.1	116.2	17.8	15.1	16.0	**14.0**	14.4	22.5

Table 6.3: SMAPE (%). Lower is better. Best per row in **bold**.

Output (Target)	Linear Reg.	LASSO	Ridge	Elastic Net	LSTM	XGBoost	Random Forest	GRU
SH-45	-1.610	0.308	-0.251	0.249	0.221	0.115	**0.325**	-0.312
SH7510	0.577	0.404	0.646	**0.656**	0.456	0.576	0.351	0.341
SH1521	-1.365	0.151	0.263	0.227	0.195	**0.426**	0.38	-0.0471
PHGD6	-15.796	-0.776	-0.624	-0.124	0.063	**0.072**	-0.11	-0.487
CHC	-6.640	-0.682	0.104	0.161	-0.039	**0.441**	-0.09	-0.498
CHO	-3.506	**-0.014**	-0.656	-0.021	-0.539	-0.059	-0.059	-0.648
Mean R^2	-4.7	-0.102	-0.086	0.19	0.06	**0.26**	0.13	-0.28

Table 6.4: R^2 score. Higher is better. Best per row in **bold**.

Output (Target)	Linear Reg.	LASSO	Ridge	Elastic Net	LSTM	XGBoost	Random Forest	GRU
SH-45	0.352	0.197	0.246	0.207	0.235	0.218	0.20	0.263
SH7510	0.493	0.122	0.12	0.119	0.146	0.109	0.15	0.161
SH1521	0.6	0.137	0.171	0.131	0.174	0.12	0.18	0.228
PHGD6	1.205	0.256	0.252	0.198	0.184	0.181	0.3	0.366
CHC	0.57	0.214	0.152	0.14	0.131	0.112	0.19	0.234
CHO	0.725	0.222	0.271	0.233	0.265	0.241	0.23	0.289
Mean nMAE	0.6574	0.1912	0.2021	0.1714	0.1891	0.1636	0.21	0.2567

Table 6.5: Normalized Mean Absolute Error (nMAE) per output. Lower is better.

Output (Target)	Linear Reg.	LASSO	Ridge	Elastic Net	LSTM	XGBoost	Random Forest	GRU
SH-45	0.41	0.247	0.332	0.257	0.313	0.279	0.23	0.324
SH7510	0.574	0.181	0.14	0.138	0.195	0.153	0.20	0.205
SH1521	0.846	0.212	0.197	0.202	0.23	0.174	0.22	0.291
PHGD6	1.339	0.329	0.314	0.261	0.232	0.237	0.36	0.424
CHC	0.754	0.291	0.213	0.206	0.21	0.168	0.29	0.344
CHO	0.936	0.283	0.362	0.284	0.329	0.289	0.29	0.361
Mean nRMSE	0.8097	0.257	0.2595	0.2246	0.2515	0.2167	0.265	0.3248

Table 6.6: Normalized Root Mean Squared Error (nRMSE) per output. Lower is better.

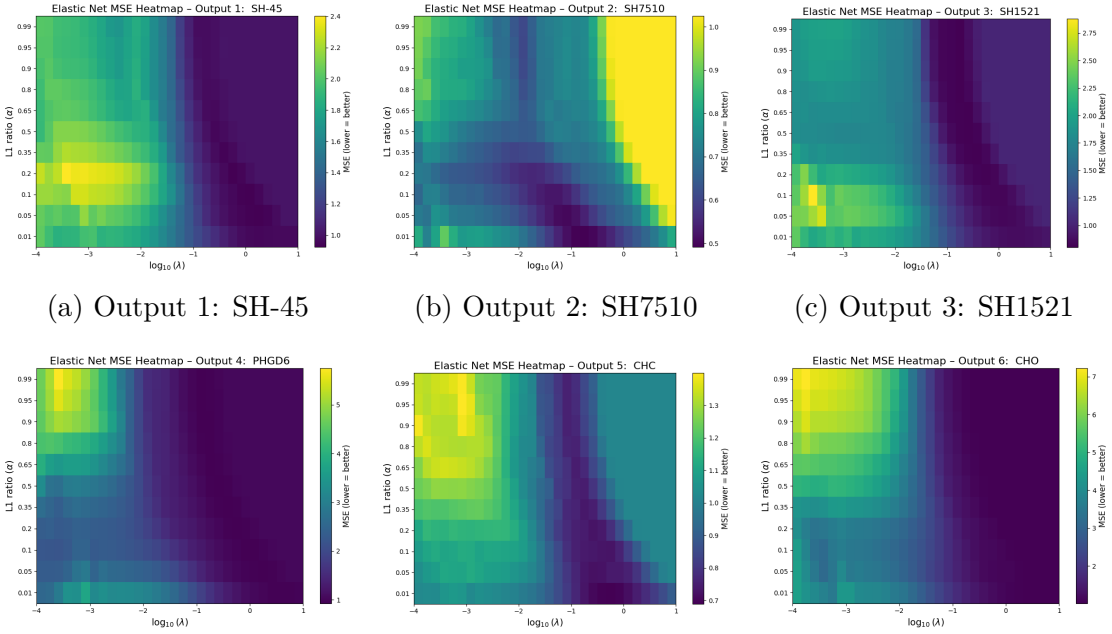
Looking at the results and different performance metrics, the mean MAE and RMSE scores, summarize each model's performance for all outputs, but they are skewed to outputs with higher magnitudes of values. The R^2 score is very dependant on the output's variance, see 3.1.1, an individual output with small variance makes this metric unreliable. Similarly, SMAPE is very sensitive for small values, which create a small denominator, and the metric becomes unstable. The key metrics to compare are the normalized nMAE and nRMSE, which show how well the model performs evenly among all outputs. Looking at the results in the tables, across the board, the most successful linear model is the elastic net, the best sequence model is the LSTM, and XGBoost edges out Random Forest for the best tree-based model. These three models will be compared and evaluated from here on.

6.2 Hyperparameter Tuning Results

The strategy for tuning hyperparameters for the three presented models are presented in this section. The strategy is determined based on the number of hyperparameters, what range of values should be tested, and how costly the computations are. When possible, visualizations are shown.

6.2.1 Elastic Net

The elastic net has two hyperparameters that are used to regulate the model, shown in equation 3.11. To optimise these two parameters, we calculate a range of separate values for λ and α, comparing the outputs based on the mean squared error. For the ℓ_1 ratio α values, we tested 11 values in the range of $\lambda \in [0.01, 0.99]$. For the regularisation strength λ we tested 30 values ranging from $\lambda \in [10^{-4}, 10]$, computed by `numpy.logspace(-4, 1, 30)`. The best combination of these two hyperparameters is found for each target variable, and a separate elastic net model is used to predict each outcome. The results of hyperparameter tuning is shown below in a heat map format, where the darker colours indicate a lower error and a better combination.



(a) Output 1: SH-45 (b) Output 2: SH7510 (c) Output 3: SH1521

(d) Output 4: PHGD6 (e) Output 5: CHC (f) Output 6: CHO

Figure 6.1: Elastic Net hyperparameter heat maps for all 6 outputs.

The resulting best combinations for each output is:

Output	**Best λ**	**Best α**
SH-45	1.2068	0.10
SH7510	0.1842	0.01
SH15121	0.1842	0.95
PHGD6	0.3728	0.35
CHC	0.5964	0.01
CHO	10.0000	0.05

Table 6.7: Optimal hyperparameters for Elastic Net for each output.

6.2.2 LSTM

The Long Short Term Memory uses a single model to predict all outputs, and thus we are aiming to find the global optimal parameter settings for all target variables. As the LSTM model consists of numerous internal settings, an initial search of multiple values for all parameters was performed. To evaluate the test performance, the mean of the root mean square errors for all 6 outputs was used.

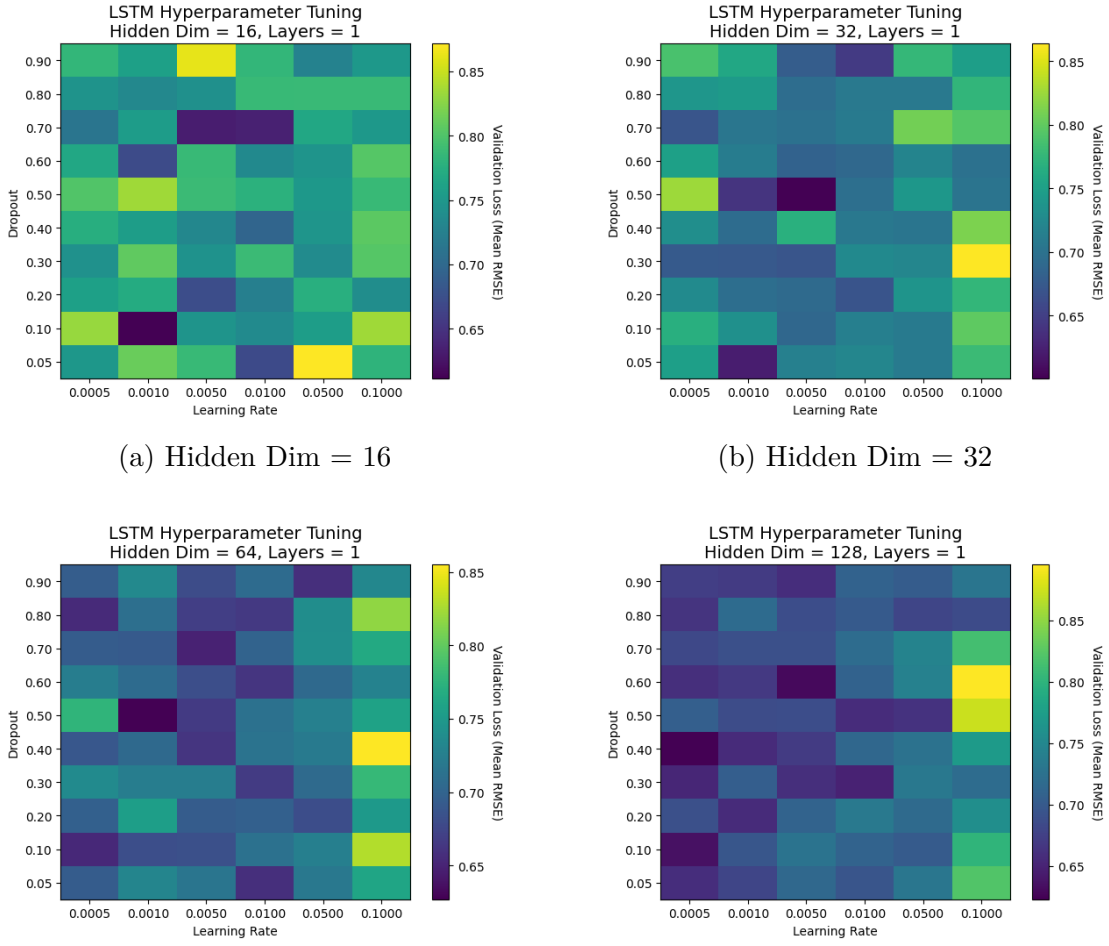
Hidden Dimension: 16, 32, 64, 128

Number of Layers: 1, 2, 3, 4

Dropout: 0.0, 0.1, 0.2, 0.5, 0.8

Learning Rate: 0.0005, 0.001, 0.003, 0.005

Sorting the results based on its test score of mean RMSE, the search did not find any particularly dominant configuration among the parameters; the only consistently superior setting was number of layers equal to 1, which appeared in 15 of the top 20 test scores. With this hyperparameter fixed, a more extensive search of values for the dropout and learning rate for a given number of layers produced the optimal combination as:



(a) Hidden Dim = 16

(b) Hidden Dim = 32

(c) Hidden Dim = 64

(d) Hidden Dim = 128

Figure 6.2: Validation loss heatmaps for different LSTM hidden dimensions.

The configurations with the lowest test score are shown below,

Hidden Dim	Layers	Dropout	Learning Rate	Validation Loss
16	1	0.10	0.001	0.6115
32	1	0.50	0.005	0.6008
64	1	0.50	0.001	0.6269
128	1	0.40	0.0005	0.6230

Table 6.8: LSTM hyperparameter configurations with validation loss by mean of RSME.

The tuning strategy and the comparisons shown in figure 6.2 show that the best performing configurations for the LSTM model is obtained with **32 Hidden Dim, 1 Hidden Layer, a Dropout rate of 0.5 and Learning Rate of 0.005**

6.2.3 XGBoost

The XGBoost consists of multiple hyperparameters; testing every single combination of a wide range of values for these variables is very demanding. A solution to this is to test random search, where we pick out a smaller range of random values for each hyperparameter. Due to the optimal settings not being very sensitive, values close together produce very similar results. This leads to a random testing of a small fraction of values, being able to find the near-optimal values in a significantly shorter time. The tuning of hyperparameters for the XGBoost model where tested in the following ranges.

Number of Estimators (n estimators): 200–600 (random integer)

Learning Rate: 0.01–0.06 (uniform)

Max Depth: 3–6 (random integer)

Subsample: 0.7–1.0 (uniform)

Colsample Bytree: 0.6–1.0 (uniform)

Regularization Alpha (reg α): 0–10 (uniform)

Regularization Lambda (reg λ): 0.1–10 (uniform)

The findings presented the optimal settings for each output for the XGBoost model as

Output	n estimators	learning rate	max depth	sub sample	colsample bytree	reg α	reg λ
SH45	298	0.010	4	0.792	0.843	9.737	2.939
SH7510	368	0.032	3	0.848	0.794	1.759	0.281
SH1521	369	0.036	5	0.770	0.609	0.467	9.838
GD	297	0.028	6	0.727	0.891	6.335	5.458
CHC	232	0.053	6	0.798	0.966	0.636	3.210
CHO	566	0.022	6	0.950	0.606	6.833	6.200

Table 6.9: Optimal hyperparameters for XGBoost for each output (rounded to 2–3 decimals).

6.3 SHAP Feature Importance Summary

In order to find the most significant features for each target variable, a SHAP analysis was performed. SHAP applies game theory to investigate each features importance to the prediction, by computing how much a prediction changes when a feature is added or removed from all possible combinations of features, see 3.1.1. A summary of the 10 most influential features according to SHAP are listed below, the extensive results are presented in the appendix **??**.

Elastic Net SHAP Analysis

Due to the elastic nets ℓ_1-norm, it can produce a sparse model, which means that many features will be shrunk to zero and removed, these features will have a SHAP score of zero. Below the features that showed a non-zero score for multiple target variables are presented.

Output	Feature	SHAP Score
SH-45	HeatZone08 > TempBot	0.108
	CoolZone02 > Dewpoint	0.061
	CoolZone04 > CoolFan	0.057
SH7510	HeatZone08 > TempTop	0.511
	HeatZone07 > TempTop	0.299
	HeatZone08 > TempBot	0.289
SH1521	CoolZone04 > CoolFan	0.229
	HeatZone06 > TempBot	0.101
	Media > DewpointCoolZone	0.090
GD	HeatZone08 > TempBot	0.148
	HeatZone07 > TempTop	0.090
	Media > NitrogenFlowWirelock	0.083
CHC	CoolZone03 > Dewpoint	0.146
	HeatZone07 > TempBot	0.121
	Media > ProcessWater(8m)	0.119
CHO	—	—

Table 6.10: Top SHAP Features per Output

The features that were present for mulitple outputs are especially interesting.

Feature	Frequency
HeatZone08 > TempBot	5
Media > DewpointCoolZone	5
HeatZone06 > TempBot	5
CoolZone04 > CoolFan	4
Media > HydrogenCoolZone	4
Input > BinWeight	4
General > Underpressure	4
HeatZone07 > TempTop	4
CoolZone02 > Dewpoint	3
HeatZone08 > TempTop	3

Table 6.11: Elastic Net top SHAP Features and Their Frequency Across Target Variables

LSTM SHAP Analysis

LSTM is not a sparse model, and for every output, the model aims to balance all 57 features. Although many features may have a very small impact, the chance of a feature being removed is unlikely, meaning all features with have a non-zero score. The SHAP analysis for LSTM is presented as the features with the highest total SHAP score, summing up the scores for each output.

Output	Feature	SHAP Score
SH-45	General>MillSpeed	0.004419
	Media>DewpointTempCoolZone	0.003581
	Media>ProcessWater(8m)	0.003238
SH7510	CoolZone03>Dewpoint	0.003539
	General>MillSpeed	0.003492
	General>MillPower	0.002847
SH1521	CoolZone04>Dewpoint	0.003113
	Media>DewpointTempCoolZone	0.002712
	CoolZone02>Dewpoint	0.002265
GD	CoolZone03>Dewpoint	0.002225
	Media>DewpointTempCoolZone	0.002064
	Input>BinWeight	0.001312
CHC	Media>DewpointTempCoolZone	0.004723
	Media>DewpointCoolZone	0.003062
	Input>BinWeight	0.002485
CHO	Media>DewpointTempCoolZone	0.004003
	General>MillSpeed	0.003688
	CoolZone02>Dewpoint	0.003246

Table 6.12: LSTM Top 3 SHAP Features per Output

Feature	Total SHAP Score
General > MillSpeed	0.0189
Media > DewpointTempCoolZone	0.0150
CoolZone04 > Dewpoint	0.0134
General > Underpressure	0.0130
General > MillPower	0.0124
Media > ProcessWater(8m)	0.0120
CoolZone03 > Dewpoint	0.0102
HeatZone01 > TempBot	0.0096
CoolZone02 > Dewpoint	0.0088
Input > BinWeight	0.0072

Table 6.13: Top 10 LSTM Features Ranked by Total SHAP Importance

XGBoost SHAP analysis

Due to XGBoost having incorporated ℓ_1 regularization, it can remove features, creating a sparse model. Although XGBoost contains its own feature importance tools, which are noted as the gain score, we will use SHAP scores for consistency among models. Below is a table that shows the three features that had the highest SHAP score per output, as well as a table that sums of the total SHAP scores among all outputs, showing the top 10 most important features for the XGBoost model.

Output	Feature	SHAP Score
SH-45	CoolZone04 > WaterTemp	0.074
	Media > DewpointCoolZone	0.036
	CoolZone02 > Dewpoint	0.023
SH7510	CoolZone02 > WaterFlow	0.220
	Media > OxygenLevelCoolZone	0.183
	CoolZone02 > Dewpoint	0.145
SH1521	Media > ProcessWater(8m)	0.116
	CoolZone04 > CoolFan	0.095
	HeatZone08 > TempBot	0.076
GD	HeatZone06 > TempTop	0.090
	HeatZone08 > TempBot	0.061
	HeatZone05 > TempBot	0.058
CHC	CoolZone01 > WaterFlow	0.185
	CoolZone04 > Dewpoint	0.130
	General > VolumeWeight	0.093
CHO	General > CakeHight	0.158
	Media > HydrogenFurnaceRear	0.088
	Media > Currentℓ_1	0.045

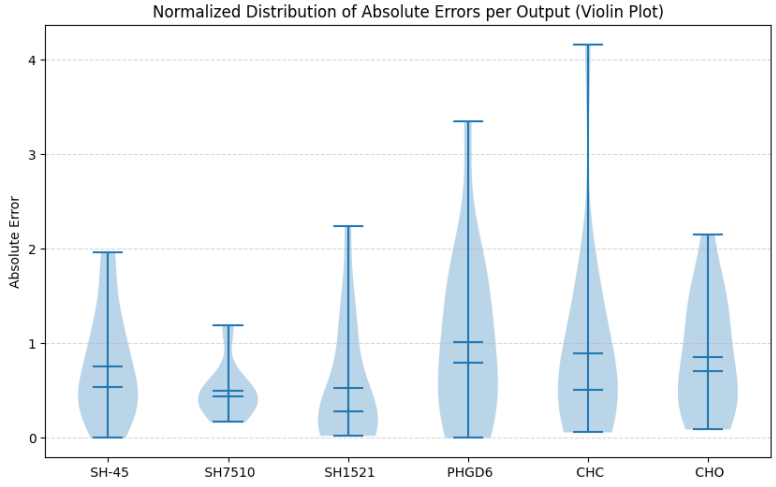
Table 6.14: XGBoost Top 3 SHAP Features per Output

Feature	Total SHAP Score
1. CoolZone02>Dewpoint	0.32241
2. CoolZone02>WaterFlow	0.29533
3. CoolZone02>CoolFan	0.24157
4. HeatZone08>TempBot	0.23842
5. CoolZone01>WaterFlow	0.23033
6. Media>OxygenLevelCoolZone	0.22324
7. CoolZone04>Dewpoint	0.21778
8. Media>DewpointCoolZone	0.21210
9. General>CakeHight	0.19893
10. CoolZone04>CoolFan	0.17329

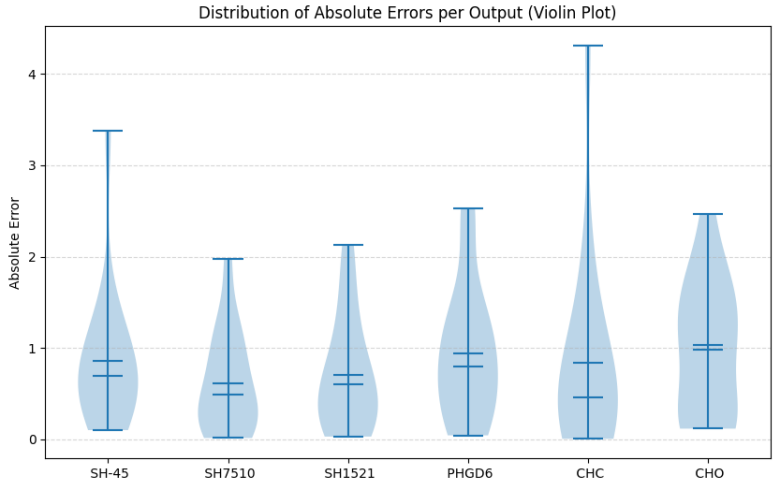
Table 6.15: Top 10 XGBoost Features Ranked by Total SHAP Importance

6.4 Violin Plots

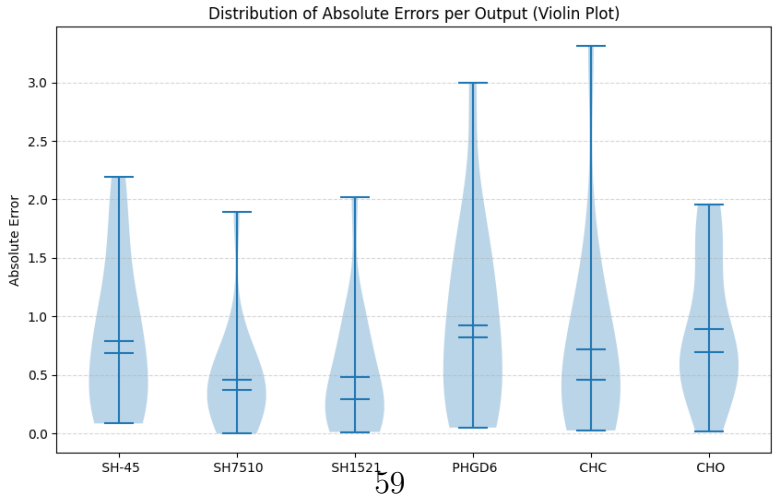
While the evaluation metric scores show quantitive values for all models for each output, it does not give an intuitive visualization of each models performance. To complement the result tables, we use violin plots to show the error distribution of each model. For each output the violin show the smallest and largest error made, and the thickness of the shape shows how often an error of this magnitude occurred. The two lines in the central parts of each shape are the mean and median errors. The following violin plots show the error for normalized data, which means no matter the differences in output sizes in magnitude, the violin plots are scale invariant.



(a) Elastic Net



(b) LSTM



(c) XGBoost

Figure 6.3: Normalized violin plot comparison across Elastic Net, LSTM, and XGBoost.

6.5 PCA Analysis

As previously mentioned in section 5.1.4, we evaluated dimensionality reduction (PCA) on the preprocessed *per-lot* and *per-timestamp* datasets. Using a explained variance target of 95%, the PCA retained:

- **Per-timestamp** (61 PV tags → 24 PCA components)
- **Per-lot** (366 descriptive statistics → 27 PCA components).

The first three principal components (PC) and its top contributors are displayed in the tables below (see Table 6.16 and 6.17), with the *loading* of a certain feature indicating how strongly that feature contributes to the direction of that certain PC. Additionally a summary of the variance-weighted contributors (Table 6.18 in done across all principal components to give a indication of the single feature importance for both datasets. Because earlier components explain more variance, we calculate the overall importance of each feature by summarizing each features squared loading over all the PCs, weighted by each PC's explained variance. This yields a single ranking per feature that reflects both how strongly it loads on each PC and how much variance those PCs explain. Higher scores therefore indicate features that are influential across the most informative PCs.

Per-Timestamp

Component (EV share)	Top contributors (|loading|)
PC1 (0.421)	HeatZone08 TempBot (0.194)
	HeatZone03 TempBot (0.193)
	HeatZone06 TempTop (0.193)
	HeatZone07 TempTop (0.193)
	HeatZone05 TempBot (0.193)
PC2 (0.118)	CoolZone01 Dewpoint (0.296)
	CoolZone02 Dewpoint (0.277)
	CoolZone03 Dewpoint (0.240)
	CoolZone04 WaterTemp (0.237)
	CoolZone03 WaterTemp (0.234)
PC3 (0.074)	CoolZone03 Dewpoint (0.314)
	Media DewpointTempCoolZone (0.312)
	Media ProcessWater(8m) (0.301)
	CoolZone04 Dewpoint (0.285)
	CoolZone02 Dewpoint (0.265)

Table 6.16: Top variables (absolute loadings) for the first three principal components in **per-timestamp** space. Magnitudes indicate contribution.

Per-Lot

Component (EV share)	Top contributors (|loading|)
PC1 (0.417)	HeatZone07 TempTop-madiff (0.0816)
	CoolZone01 WaterTemp-median (0.0814)
	HeatZone01 TempTop-std (0.0814)
	HeatZone08 TempBot-min (0.0813)
	HeatZone07 TempBot-madiff (0.0812)
PC2 (0.113)	CoolZone01 Dewpoint-mean (0.1229)
	CoolZone01 Dewpoint-min (0.1216)
	CoolZone02 Dewpoint-mean (0.1212)
	CoolZone02 Dewpoint-min (0.1206)
	Media DewpointCoolZone-mean (0.1184)
PC3 (0.071)	General RotatingFeeder-max (0.1530)
	Media CO-MeasureRear-madiff (0.1520)
	CoolZone03 CoolFan-max (0.1402)
	Output OutletWeight-min (0.1369)
	Media DewpointTempRear-max (0.1326)

Table 6.17: Top variables (absolute loadings) for the first three principal components in **per-lot** space. Magnitudes indicate contribution.

Per-timestamp	Per-lot
Feature (score)	**Feature (score)**
Input BinWeight (0.0167)	HeatZone04 TempTop-median (0.0028)
Output OutletWeight (0.0167)	CoolZone04 CoolFan-median (0.0028)
Output OutletbinWeight (0.0167)	HeatZone05 TempTop-mean(0.0028)
General CakeHight (0.0167)	HeatZone05 TempTop-median (0.0028)
CoolZone03 WaterFlow (0.0166)	HeatZone04 TempTop-mean (0.0028)
Media OxygenLevelCoolZone (0.0166)	HeatZone07 TempTop-mean (0.0028)
HeatZone07 TempTop (0.0166)	HeatZone04 TempBot-median (0.0028)
HeatZone04 TempTop (0.0166)	HeatZone03 TempBot-median (0.0028)
HeatZone06 TempTop (0.0166)	HeatZone07 TempTop-median (0.0028)
HeatZone05 TempTop(0.0166)	HeatZone05 TempBot-median(0.0028)
HeatZone05 TempBot (0.0166)	CoolZone03 CoolFan-min (0.0028)
HeatZone03 TempBot (0.0166)	CoolZone02 CoolFan-std (0.0028)
HeatZone04 TempBot (0.0166)	CoolZone03 CoolFan-madiff (0.0028)
HeatZone06 TempBot (0.0166)	CoolZone02 CoolFan-madiff (0.0028)
HeatZone07 TempBot (0.0166)	CoolZone03 CoolFan-std (0.0028)

Table 6.18: Global variance-weighted contributors across all PCs — per-timestamp vs per-lot

The large reduction in features to principal components, for both datasets, indicate a strong correlation between the features in the original datasets. This is of course to be expected as many of the measured signals from the real-life process are heavily correlated (such as temperature measurements in heat zones). This can be seen by the top contributors of the PCs where, for example, PC1 from both datasets has many HeatZone components and PC2 consists of features concerning the CoolZones. From the global variance-weighted contributions in Table 6.18 we note that the temperature measurements from the heat zones are important features in both datasets and the per-lot datasets prioritizes CoolZone features whereas the per-timestamp dataset emphasizes features such as weight. Another conclusion we can draw from the per-lot column in Table 6.18 is that median and mean features seem to be more significant that other descriptions. Noteworthy however is that the score ranking the contributions indicates that the difference in weighted contribution is minimal ($\frac{1}{61} \approx 0.0164$ and $\frac{1}{366} \approx 0.0027$) and thus the results of the PCA analysis does not seem to be of large significance as most features are moderately equally weighted, which is one reason why the dimensionally reduced PCA datasets where not used during the final model evaluations.

Chapter 7

Results and Discussion

In this section results of the best performing models, one from each of the "model families" (linear, ensemble/tree based and sequence models) are discussed and compared.

7.1 Elastic Net

The elastic net is a strong-performing model for the annealing process. Its combined ℓ_1 and ℓ_2 penalties allow it to remove unimportant features and also group together collinear features, shrinking them together, which keeps them in the model. The hyperparameter tuning for each output clearly shows how α and λ impact each other, with the existence of favourable ranges and areas for which we receive a lower test score. This allows us to make some assumptions of the analysis data we are predicting.

Output 1, which is predicting the weight percent of the smallest particles SH-45, and especially output 6, which is the weight percent of oxygen in the metal powder, benefit from having a higher α value. This leads to a higher regularization, and the model becomes more aggressive in its shrinking and removal of features. This is helpful for data that is either very noisy or has a high variance. Outputs 2-5, all have moderately small α values, suggesting the data here is clear and should not be regularized. A lower α means the model behaves more closely to a linear regression, suggesting there are more independent features contributing to predicting the outcome.

The values of the α, which determines if the elastic net should lean more towards a LASSO (removing features) or ridge (shrinking). Outputs 1, 2 , 5, 6 all showed low values as optimal after the tuning, which results in a more ridge-like model, suggesting many features may be collinear but are all helpful and meaningful. Output 3, which is the weight percent distribution of the larger particle size

SH1521 had the optimal ℓ_1 ratio score of $\alpha = 0.95$, which sets the elastic net very close to a LASSO model. For predicting this output, there exist only a few strong predictors among the features; the optimal strategy in this scenario is to remove features that are not meaningful, which ℓ_1-regularization does swiftly. Output 4, which is the chemical weight percent of carbon, benefited from a mix between both models.

The SHAP analysis presented 3 features that received a non-zero score for 5 out of 6 outputs. These were `HeatZone08>TempBot`, `Media>DewpointCoolZone`, and `HeatZone06>TempBot`. There were 4 more features that appeared in 4 out of 6 outputs, and many more appeared in 3. Curiously, the SHAP analysis did not select a single feature for the last output. This connects well to expectations, considering the optimal α value was 10, which was the highest value tested. The elastic net also presented a negative R^2 score for this output, meaning the model did not find any patterns for this particular outcome and predicted worse than the mean of the data. It is clear that the elastic net struggles with this output. Comparing the evaluation metrics to other models, the elastic net outperforms both the ℓ_1, and ℓ_2 regularization for almost all outputs and all metrics, this a sign of good hyperparameter tuning. The target variables where the elastic net produced the best performance were for outputs 1 and output 2, where it generally achieved better scores than both LSTM and XGBoost/Random Forest.

Prediction for random test lot

Output	True Value	Predicted Value	Absolute Error
SH45	15.100	15.5474	0.4474
SH7510	28.100	29.1361	1.0361
SH1521	8.300	7.1036	1.1964
GD	6.960	6.9517	0.0083
CHC	0.002	0.0014	0.0006
CHO	0.143	0.1472	0.0042
		Mean Absolute Error	**0.4488**

Table 7.1: Predictions by the Elastic Net model for a random test lot.

7.2 LSTM

Looking at the LSTM model's performance, although it is regularly in the top ranks, it strangely doesn't achieve the lowest score for any of the evaluation metrics. Comparing it to GRU, the difference is marginal for mean absolute error, but

is considerably better for root mean square error, suggesting the LSTM handles outliers better than its simpler counterpart. Experiments showed that having a single LSTM model for each output was not beneficial for a singular model, allowing it to learn patterns between outputs as well. Looking at the hyperparameter tuning 6.2 reveals a high degree of randomness within the LSTM. This comes from the initial weights in the neural networks being random; they are improved using the backpropagation algorithm and may converge to similar values, but this is not guaranteed. The only hyperparameter setting that was clearly best was keeping the layers to 1.

The dropout value is a regularization technique that randomly removes some nodes in the LSTM's neural network to prevent overfitting; however this does not apply for the last layer. This means that for the case of only having 1 layer, the dropout rate is essentially not doing anything, this is clearly seen in the figures as there are no trends indicating that the dropout rate is significant to the 1 layer LSTM model. The learning rate seems to have a small impact, figure c) and figure d) with higher hidden dimensions, show weak signs of a low learning rate being preferred. The case of neither layer dim, dropout rate or learning rate having a large impact on the test scores show that the single most important hyperparameter setting was to set the number of layers to 1.

Looking at the SHAP analysis, due to the non-sparsity we had many contributing features for each output. Summing up the scores over all six outputs gives us our ten most central features, in the top 3 we find `General>MillSpeed, Media>DewpointTempCoolZone, and CoolZone04>Dewpoint`. This difference is quite substantial but their similar test scores suggest that the way linear and sequence model utilize data in their predictions are different but equally effective.

Output	**True Value**	**Predicted Value**	**Absolute Error**
SH45	15.1000	14.6208	0.4792
SH7510	28.1000	29.0804	0.9804
SH1521	8.3000	7.9151	0.3849
GD	6.9600	6.9583	0.0017
CHC	0.0020	0.0015	0.0005
CHO	0.1430	0.1466	0.0036
	Mean Absolute Error		**0.3084**

Table 7.2: Prediction by the LSTM model for a random test lot.

7.3 XGBoost

XGBoost performed very well on all evaluation metrics, achieving the lowest score out of all models for the nMAE and the joint lowest for nRMSE, tied with random forest. XGBoost performs especially well for data with little noise and variance, shown by its lowest nMAE score. Either XGBoost handles data with more outliers worse or it could be its tree counterpart, random forest that does extra well, shown by their nRSME scores.

Due to the model having a large number of hyperparameters, the tuning of internal settings was not as simple as for the other models. The testing showed that some hyperparameters all found similar values for all outputs such as the subsample size or colsample_bytree, shown in **??**, while others presented a range of different values, for example the regularization lambda and alpha. These may need extra tuning to find the true optimal values.

Looking at the SHAP results, we find some patterns for each output. The size distributions, which are the first 3 outputs, all show similar features, while the last 3 show more unique features. The combined SHAP scores show CoolZone2 as being the most vital part of the process.

Prediction of random test lot

Output	True Value	Predicted Value	Absolute Error
SH45	15.9000	14.5204	1.3796
SH7510	27.3000	28.0396	0.7396
SH1521	7.7000	7.8245	0.1245
GD	6.9500	6.9517	0.0017
CHC	0.0012	0.0018	0.0006
CHO	0.1780	0.1451	0.0329
		Mean Absolute Error	**0.3798**

Table 7.3: Predictions by the XGBoost model for a random test lot.

7.4 Comparisons

All models fail to achieve a positive R^2 score for the last output. This is a very curious result, as the other metrics do not indicate any substantially worse performances for this particular output. The common cause for this would be the R^2 score being dependent on the output's variance, see 3.1.1. An output with low variance or a lot of noise, may result in a low R^2 score, but this does not necessarily

indicate that the model is performing badly. The table 4.2 shows that output 6, or CHO, has the joint lowest normalized std together with output 1. This, together with the fact that this output also has a considerably low mean compared to the other outputs, may be the reason why this metric is negative for all 3 models.

The violin plots for all three models show similar trends. The lowest absolute errors, corresponding to the height of each violin shape, was for output 2, which was the SH7510 size distribution. The highest absolute error was for output 5, which was CHC, a carbon weight percent measure of the powder.

Chapter 8

Conclusion and Future Work

The conclusions we can deduce from this thesis are presented in a short summary, while we also suggest improvements and future work that could be made.

8.1 Conclusion

We have implemented eight separate machine learning models and evaluated their performances on the data given to us. The wide range of performance metrics presented combine to give a robust foundation to draw conclusions from.

Something that can be remarked upon being perhaps unexpected, is that all three models perform almost identically over all outputs, with only marginal differences pushing either model in front of the other. Despite the fundamental differences in how the elastic net, LSTM, and XGBoost work, training the models on the same dataset, leads them to arrive at the same conclusions. This is evident in the performance metric tables as well as the violin plots. The SHAP analysis presents the features that are most influential for the model's predictions. This thesis is a benchmark and initial investigation for machine learning's potential within powder metallurgy processes and highlights its strengths and where improvements must be made. In conclusion, we can say that our thesis has succeeded to some degree with our purpose with many suggestions for future work presented in the next section.

8.2 Future Work

There are a number of improvements one could make to further expand and improve on the results found in this thesis. A straightforward improvement would be to increase the available data volume and to improve the alignment between the recorded data and the physical annealing process it represents. During this thesis,

the production- and analysis data were linked together using the lot key as an identifier, as described in Section 5.1. Referring back to Section 4.3, we note that one production lot covers the time span of around $\sim$ 20 hours in the annealing furnaces. A natural next step is to utilize the timestamps from both work centre exports to implement a *time-resolved* linkage, instead of using the *lot-based* linkage between the features in the production data to the targets in the analysis data. The analysis data contains a time column named `labreceived` which indicates the exact timestamp at which a sample arrives at a lab and is registered by a lab technical for analyses. In consultation with Höganäs, an estimate for the time it takes from sample collection completion to when a lab technician registers the sample (*time_lag*) could be made. Since we know that the total time it takes for a sample to be collected is around 4 hours (as mentioned in Section 4.3) we could instead use the `labreceived` timestamps to link the analyses to the production data in the time interval. Let

$t_{\text{lab}} :=$ `labreceived`, $\quad \tau_{\text{lag}} :=$ expected delay from sampling completion to registration

$$\tau_{\text{sample}} \approx 4h.$$

Then the production interval that most directly "causes" the measured quality can be approximated as

$$\boxed{I_{\text{sample}} \;=\; \Big[t_{\text{lab}} - \tau_{\text{lag}} - \tau_{\text{sample}},\; t_{\text{lab}} - \tau_{\text{lag}}\Big]} \tag{8.1}$$

Optionally, a small slack δ (e.g., $\pm$30 min) can be applied to the bounds to account for operator variability and queueing.
This modification would likely improve the results since now the feature extraction is narrowed down to the few hours that directly correspond to the sampled powder, rather that the entire $\sim$20 h lot. Focusing on the more narrow time interval of I_{sample} would remove unrelated time slices from the whole lot window, reducing dilution of the feature aggregation and thus tightening the relation between features and targets.

This modification would also possibly increase the *Number of Samples* challenge discussed in 4.4. The study could also be expanded to include the setpoints (SP) from the features or use these features instead of the process values (PV). The setpoints better represent to the actual values which the operators can modify and tweak on their machines but may have the downside that they have a worse correlation with the target values, as they do not account for discrepancies, and thus may produce even worse results for ML the ML models trying to predict the outcome.

Bibliography

[1] J. Tengzelius, Y. Axelsson, K. Fernheden, P. Hasselström, C. Karlsson, and B. Uhrenius, *Powder Metallurgy in Sweden - 100 Years of Development*, ser. Jernkontorets bergshistoriska skriftserie. Jernkontoret, 2022.

[2] Höganäs AB (publ.), "Astaloy crm," Höganäs AB, Höganäs, Sweden, Product datasheet 2076HOG, Oct. 2017. [Online]. Available: https://www.hoganas.com/globalassets/downloads/libary/astaloy_astaloy-crm_2076hog.pdf

[3] Höganäs AB (publ.), "Astaloy cra — datasheet." [Online]. Available: https://www.hoganas.com/globalassets/downloads/libary/astaloy_astaloy-cra_1541hog.pdf

[4] ——. Astaloy cra | höganäs. Product page. [Online]. Available: https://www.hoganas.com/en/powder-technologies/products/astaloy/astaloy-cra/

[5] Höganäs AB (publ.), "Höganäs handbook for sintered components: Volume 2, production of sintered components," Höganäs AB, Tech. Rep., 2103, publication no. 0675HOG. [Online]. Available: /mnt/data/handbook-2_production_of_sintered_components_0675hog.pdf

[6] A. Muntin, P. Zhikharev, A. Zinyagin, and D. Brayko, "Artificial intelligence and machine learning in metallurgy. part 1. methods and algorithms," *Metallurgist*, vol. 67, pp. 124–130, 11 2023.

[7] ——, "Artificial intelligence and machine learning in metallurgy. part 2. application examples," *Metallurgist*, vol. 67, pp. 99–111, 03 2023.

[8] V. Yurov, S. Kuzenkov, and I. Tsyganov, "Application of combined machine learning methods in metallurgy for analysis, selection of materials and prediction of properties," in *2024 6th International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA)*, 2024, pp. 784–788.

[9] S.-M. Chang, H. P. Lin, C.-C. Han, and Y.-C. Wu, "An iot-based parameter extraction platform for powder metallurgy sintering furnace," in *2024 11th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, 2024, pp. 19–24.

[10] L. Mazzei and V. Ramjattan, "A review of machine learning for industry 4.0: Algorithms, applications and challenges," *Sensors*, vol. 22, no. 22, p. 8641, 2022.

[11] Y. S. Perera, D. A. A. C. Ratnaweera, C. H. Dasanayaka, and C. Abeykoon, "The role of artificial intelligence-driven soft sensors in advanced sustainable process industries: A critical review," *Engineering Applications of Artificial Intelligence*, vol. 121, p. 105988, 2023.

[12] D. Massimo, E. Ganthaler, A. Buriro, F. Barile, M. Moraschini, A. Dignös, T. Villgrattner, A. Peer, and F. Ricci, "Estimation of mass and lengths of sintered workpieces using machine learning models," *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1–14, 2023.

[13] C.-J. Chen, F.-I. Chou, and J.-H. Chou, "Temperature prediction for reheating furnace by gated recurrent unit approach," *IEEE Access*, vol. 10, pp. 33 362–33 369, 2022.

[14] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[15] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, "A review of process fault detection and diagnosis. part i: Quantitative model-based methods," *Computers & Chemical Engineering*, vol. 27, no. 3, pp. 293–311, 2003.

[16] ——, "A review of process fault detection and diagnosis. part ii: Qualitative models and search strategies," *Computers & Chemical Engineering*, vol. 27, no. 3, pp. 313–326, 2003.

[17] ——, "A review of process fault detection and diagnosis. part iii: Process history based methods," *Computers & Chemical Engineering*, vol. 27, no. 3, pp. 327–346, 2003.

[18] Y. Cai, R. Li, Y. Chen, and W. Zhang, "Stacked ensemble learning model-based prediction and optimization of the grade of titanium dioxide in high-titanium slag," *Journal of Sustainable Metallurgy*, 2025.

[19] C. Cui, G. Cao, X. Li, Z. Gao, X. Zhang, and Z. Zhou, "A strategy combining machine learning and physical metallurgical principles to predict mechanical properties for hot rolled ti micro-alloyed steels," *Journal of Materials Processing Technology*, vol. 311, p. 117810, 2023.

[20] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

[21] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York: Springer, 2009. [Online]. Available: https://doi.org/10.1007/978-0-387-84858-7

[22] "Model fit: Underfitting vs. overfitting," https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html, n.d., accessed: 2025-12-09.

[23] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207006000239

[24] L. Antwarg, R. M. Miller, B. Shapira, and L. Rokach, "Explaining anomalies detected by autoencoders using shapley additive explanations," *Expert Systems with Applications*, vol. 186, p. 115736, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421011155

[25] Y. Nohara, K. Matsumoto, H. Soejima, and N. Nakashima, "Explanation of machine learning models using shapley additive explanation and application for real data in hospital," *Computer Methods and Programs in Biomedicine*, vol. 214, p. 106584, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169260721006581

[26] V. Nasteski, "An overview of the supervised machine learning methods," *Horizons. b*, vol. 4, no. 51-62, p. 56, 2017.

[27] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC4792409/

[28] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4,

pp. 237–285, 1996, accessed: 2025-11-25. [Online]. Available: https://www.jair.org/index.php/jair/article/view/10166

[29] J. Euchner, "Generative ai," *Research-Technology Management*, vol. 66, no. 3, pp. 71–74, 2023.

[30] F. F.-H. Nah, R. Zheng, J. Cai, K. Siau, and L. Chen, "Generative ai and chatgpt: Applications, challenges, and ai-human collaboration," *Journal of Information Technology Case and Application Research*, vol. 25, no. 3, pp. 277–304, 2023.

[31] J. Shurman, "Multivariable calculus," https://www2.stat.duke.edu/~sayan/informal/vcalc.pdf, 2015, reed College. Accessed: 2025-11-25.

[32] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016, version v2, last revised 15 Jun 2017. [Online]. Available: https://arxiv.org/abs/1609.04747

[33] I. Gitman, H. Lang, P. Zhang, and L. Xiao, "Understanding the role of momentum in stochastic gradient methods," in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019, paper ID 5108. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/4eff0720836a198b6174eecf02cbfdbf-Paper.pdf

[34] C. Multiphysics. (2023) Deep neural network. COMSOL AB. Accessed: 2025-11-25. [Online]. Available: https://doc.comsol.com/6.2/doc/com.comsol.help.comsol/comsol_ref_definitions.19.050.html

[35] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Science*, vol. 6, no. 12, pp. 310–316, 2017.

[36] M. Cilimkovic, "Neural networks and back propagation algorithm," Institute of Technology Blanchardstown, Blanchardstown Road North, Dublin 15, Technical Report 1, 2015. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/51

[37] T. M. Hope, "Chapter 4 - linear regression," in *Machine Learning*, A. Mechelli and S. Vieira, Eds. Academic Press, 2020, pp. 67–81. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128157398000043

[38] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 67, no. 2, pp. 301–320, 03 2005. [Online]. Available: https://doi.org/10.1111/j.1467-9868.2005.00503.x

[39] G. C. McDonald, "Ridge regression," *WIREs Computational Statistics*, vol. 1, no. 1, pp. 93–100, 2009. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/wics.14

[40] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970. [Online]. Available: https://doi.org/10.1080/00401706.1970.10488634

[41] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996. [Online]. Available: https://webdoc.agsci.colostate.edu/koontz/arec-econ535/papers/Tibshirani%20(JRSS-B%201996).pdf

[42] W. J. Fu, "Penalized regressions: The bridge versus the lasso," *Journal of Computational and Graphical Statistics*, vol. 7, no. 3, pp. 397–416, 1998. [Online]. Available: http://www.jstor.org/stable/1390712

[43] E. Halabaku and E. Bytyçi, "Overfitting in machine learning: A comparative analysis of decision trees and random forests." *Intelligent Automation & Soft Computing*, vol. 39, no. 6, 2024.

[44] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 10 2001.

[45] M. Niazkar, A. Menapace, B. Brentan, R. Piraei, D. Jimenez, P. Dhawan, and M. Righetti, "Applications of xgboost in water resources engineering: A systematic literature review (dec 2018–may 2023)," *Environmental Modelling Software*, vol. 174, p. 105971, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S136481522400032X

[46] XGBoost Developers. (2023) Xgboost documentation: Model architecture and parameters. Accessed: 2025-03-22. [Online]. Available: https://xgboost.readthedocs.io/

[47] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou *et al.*, "Xgboost: extreme gradient boosting," *R package version 0.4-2*, vol. 1, no. 4, pp. 1–4, 2015.

[48] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[49] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*. Pmlr, 2013, pp. 1310–1318.

[50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[51] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

Appendix A

SHAP Feature Importance

Elastic Net SHAP scores

Output	Feature Group	SHAP score
1	HeatZone08 > TempBot	0.108
	CoolZone02 > Dewpoint	0.061
	CoolZone04 > CoolFan	0.057
	Media > DewpointCoolZone	0.055
	HeatZone08 > TempTop	0.043
	Media > HydrogenCoolZone	0.036
	HeatZone06 > TempTop	0.032
	Media > ProcessWater (8m)	0.030
	HeatZone06 > TempBot	0.030
2	HeatZone08 > TempTop	0.511
	HeatZone07 > TempTop	0.299
	HeatZone08 > TempBot	0.289
	HeatZone05 > TempBot	0.199
	General > Underpressure	0.188
	CoolZone02 > Dewpoint	0.183
	HeatZone07 > TempBot	0.182
	Media > OxygenLevelCoolZone	0.177
	Media > HydrogenCoolZone	0.175
3	CoolZone04 > CoolFan	0.229
	HeatZone06 > TempBot	0.101
	Media > DewpointCoolZone	0.090
	HeatZone08 > TempBot	0.068
	Input > BinWeight	0.014
4	HeatZone08 > TempBot	0.148
	HeatZone07 > TempTop	0.090
	Media > NitrogenFlowWirelock	0.083
	HeatZone04 > TempBot	0.072
	General > Underpressure	0.070
	Media > HydrogenCoolZone	0.055
	CoolZone03 > Dewpoint	0.036
	HeatZone05 > TempTop	0.018
	CoolZone03 > WaterTemp	0.018
5	CoolZone03 > Dewpoint	0.146
	HeatZone07 > TempBot	0.121
	Media > ProcessWater (8m)	0.119
	Media > DewpointCoolZone	0.118
	HeatZone08 > TempTop	0.112
	HeatZone05 > TempTop	0.100
	Media > Currentℓ_1	0.099
	CoolZone01 > WaterFlow	0.098
	Media > NitrogenFlowWirelock	0.096
6	–	–

Table A.1: Top 10 SHAP feature groups per output for Elastic Net.

LSTM SHAP scores

Output	Feature Group	SHAP score
1	General > MillSpeed	0.003
	Media > DewpointTempCoolZone	0.003
	Media > ProcessWater (8m)	0.002
	General > Underpressure	0.002
	CoolZone03 > Dewpoint	0.002
	General > MillPower	0.002
	CoolZone04 > Dewpoint	0.002
	HeatZone01 > TempBot	0.001
	CoolZone01 > Dewpoint	0.001
2	CoolZone04 > Dewpoint	0.005
	General > MillSpeed	0.004
	Media > DewpointTempCoolZone	0.004
	CoolZone02 > Dewpoint	0.003
	General > Underpressure	0.003
	HeatZone01 > TempBot	0.003
	Media > ProcessWater (8m)	0.003
	General > MillPower	0.003
	CoolZone03 > Dewpoint	0.003
3	General > Underpressure	0.003
	Media > ProcessWater (8m)	0.003
	CoolZone04 > Dewpoint	0.003
	General > MillPower	0.002
	CoolZone02 > Dewpoint	0.002
	CoolZone01 > Dewpoint	0.002
	HeatZone01 > TempBot	0.002
	General > MillSpeed	0.002
	General > VolumeWeight	0.002
4	General > MillSpeed	0.004
	Media > DewpointTempCoolZone	0.003
	General > MillPower	0.002
	General > Underpressure	0.001
	CoolZone02 > WaterFlow	0.001
	CoolZone01 > Dewpoint	0.001
	CoolZone03 > Dewpoint	0.001
	General > VolumeWeight	0.001
	Media > ProcessWater (8m)	0.001
5	General > MillSpeed	0.005
	Media > DewpointTempCoolZone	0.003
	General > MillPower	0.003
	Input > BinWeight	0.002
	Media > ProcessWater (8m)	0.002
	CoolZone04 > Dewpoint	0.002
	Media > DewpointCoolZone	0.002
	HeatZone01 > TempBot	0.002
	Media > CO-MeasureRear	0.002
6	CoolZone03 > Dewpoint	0.002
	CoolZone04 > Dewpoint	0.002
	Input > BinWeight	0.001
	Media > DewpointTempCoolZone	0.001
	General > MillPower	0.001
	General > MillSpeed	0.001
	General > Underpressure	0.001
	CoolZone02 > Dewpoint	0.001
	CoolZone02 > WaterFlow	0.001

Table A.2: Top 10 SHAP feature groups per output for LSTM.

XGBoost SHAP scores

Output	Feature Group	SHAP score
1	CoolZone04 > WaterTemp	0.074
	Media > DewpointCoolZone	0.036
	CoolZone02 > Dewpoint	0.023
	HeatZone02 > TempBot	0.021
	CoolZone04 > CoolFan	0.021
	HeatZone06 > TempBot	0.019
	CoolZone02 > WaterTemp	0.018
	Media > NitrogenFlowWirelock	0.015
	CoolZone03 > WaterTemp	0.013
2	CoolZone02 > WaterFlow	0.220
	Media > OxygenLevelCoolZone	0.183
	CoolZone02 > Dewpoint	0.145
	HeatZone08 > TempTop	0.078
	CoolZone02 > CoolFan	0.068
	Media > CurrentL3	0.067
	Media > HydrogenFurnaceRear	0.057
	CoolZone03 > CoolFan	0.049
	Media > FurTotalPower	0.046
3	Media > ProcessWater (8m)	0.116
	CoolZone04 > CoolFan	0.095
	HeatZone08 > TempBot	0.076
	Media > DewpointCoolZone	0.072
	CoolZone02 > CoolFan	0.067
	HeatZone06 > TempBot	0.067
	CoolZone02 > Dewpoint	0.058
	CoolZone02 > WaterTemp	0.049
	Media > CurrentL3	0.045
4	HeatZone06 > TempTop	0.090
	HeatZone08 > TempBot	0.061
	HeatZone05 > TempBot	0.058
	General > BeltForce	0.043
	HeatZone08 > TempTop	0.042
	CoolZone02 > CoolFan	0.036
	CoolZone02 > Dewpoint	0.036
	General > MillPower	0.035
	HeatZone02 > TempBot	0.034
5	CoolZone01 > WaterFlow	0.185
	CoolZone04 > Dewpoint	0.130
	General > VolumeWeight	0.093
	Media > ProcessWater (12m)	0.088
	Media > Currentℓ_2	0.067
	Media > DewpointCoolZone	0.065
	Media > DewpointTempRear	0.050
	HeatZone08 > TempBot	0.045
	CoolZone01 > Dewpoint	0.041
6	General > CakeHeight	0.158
	Media > HydrogenFurnaceRear	0.088
	Media > Currentℓ_1	0.045
	CoolZone02 > CoolFan	0.039
	Media > NitrogenFlowWirelock	0.039
	CoolZone02 > Dewpoint	0.034
	HeatZone04 > TempBot	0.034
	Media > HydrogenCoolZone	0.027
	General > MillSpeed	0.027

Table A.3: Top 10 SHAP feature groups per output for XGBoost.

Master's Theses in Mathematical Sciences 2026:E6
ISSN 1404-6342

LUTFMA-3606-2026

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/