CHALMERS
UNIVERSITY OF TECHNOLOGY

Machine Learning for Force Simulation and Material Identification

Caroline Andersson

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Machine Learning for Force Simulation and Material Identification

Caroline Andersson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Division name
Chalmers University of Technology
Gothenburg, Sweden 2025

Machine Learning for Force Simulation and Material Identification.

Caroline Andersson

Supervisor: Johannes Quist, Fraunhofer Chalmers Centre
Examiner: Mattias Wahde, Mechanics and Maritime Sciences

Master's Thesis 2025
Department of Mechanics and Maritime Sciences
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Real-life picture of the experimental rig during an audio capture calibration. The components of the rig are a UR5e robot, a microphone, a plastic box and a particle bed.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

An Exploration of Machine Learning Based Solutions in Relation to Wheel Loading Operations.

Caroline Andersson
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

Virtual product development using numerical simulations is crucial for heavy machinery design. This research focuses on using machine learning models to increase the efficiency of granular loading simulations. In addition, the machine learning models are trained with data from a miniature wheel loader rig. The models predict interaction forces and classify materials using sound and vibration data, offering a robust alternative to vision-based methods in challenging conditions. To improve transparency, interpretable models are compared with deep neural networks. The conclusion is that force predictions made by a regression-based machine learning model in particular show the most similarity to DEM-based force predictions. In addition, nearly all models achieved a test accuracy of over 90% during particle classification, with the perceptron and linear kernel SVM providing the highest accuracy with the shortest training times. Lastly, a conclusion cannot be made as to whether a deep or interpretable model is better suited for the force prediction task. However, either feedback-based force generation is unsuitable or it has to be tweaked more to produce better results.

Keywords: ML, GRU, DEM, loader, granular, prediction, classification, particles, sound, force

Acknowledgements

I would like to thank everyone who has supported me throughout this project, especially my supervisor Johannes Quist who has helped me a great deal in all sorts of areas. From the FCC side, I would like to extend my thanks to Anita Ullrich for providing insights into Demify, and with her assistance as a code reviewer. Pontus Malmsköld was also a great asset for discussing general topics regarding the thesis. In addition, Daniel Gleeson provided help with the UR-robot as well as how robot path planning works in FCC's own simulation software. I would also like to thank my examiner Mattias Wahde who provided valuable feedback throughout the project.

Finally, thank you to all my friends and family who have supported me during this time. I look forward to finishing the thesis and starting a new chapter in my life as a working adult at Fraunhofer Chalmers Centre.

Caroline Andersson, Gothenburg, February 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

CAD	computer-aided design
DEM	discrete element method
GRU	gated recurrent unit
LDA	linear discriminant analysis
PCA	principal component analysis
RNN	recurrent neural network
RTDE	real-time data exchange
SVM	support vector machine
UR5e	universal robots 5e

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

Symbol	Description
W	Weight matrix used in Perceptron and other neural models.
U	Weight matrix used in GRU model.
b	Bias term in Perceptron, SVM, and GRU.
C	Regularization hyperparameter in SVM.
$\Phi(x_i)$	Nonlinear mapping function in SVM.
σ	Sigmoid activation function.
tanh	Hyperbolic tangent activation function.
α	Scaling factor in sigmoid kernel.
c	Offset term in polynomial or sigmoid kernels.
d	Degree of polynomial kernel.
σ^2	Variance in Radial Basis Function (RBF) kernel.

Variables

Symbol	Description
x	Input vector or data sample for models.
$\hat{y}$	Predicted class in Perceptron.
z_k	Score for class k in Perceptron.
y_i	Class label for sample i in SVM.
ξ_i	Slack variable in SVM for handling soft margins.
α_i	Lagrange multiplier in SVM dual formulation.

Symbol	Description
$K(x_i, x_j)$	Kernel function in SVM.
h_t	Hidden state in GRU at time t.
z_t	Update gate in GRU.
m_i	Mass of particle i in DEM.
$\mathbf{v}_i$	Velocity of particle i in DEM.
ω_i	Rotational velocity of particle i in DEM.
$\mathbf{I}_i$	Moment of inertia of particle i in DEM.
$\mathbf{F}_{ij}^c$	Force applied to particle i by collision with particle j in DEM.
$\mathbf{F}_i^g$	Gravitational force on particle i in DEM.
$\mathbf{M}_{ij}$	Torque applied to particle i by particle j in DEM.

Contents

1
Introduction

1.1 Background

Virtual product development based on numerical system simulations has become one of the most important tools in heavy machinery and vehicle development. Agricultural and construction machinery developers increasingly rely on virtual methods and digital twins. To evaluate the design performance of wheel loaders or a new concept for a harvesting machine, it is not sufficient to only create a suitable machine model. The model must also incorporate driver and operator models and accurately represent interactions with the environment, which requires detailed ways to describe these interactions [15].

(a) Demify® simulated model **(b)** A test conducted at Volvo CE

Figure 1.1: A wheel-loading scenario and its digital twin.

This interaction involves complex physical processes, such as the prediction of interaction forces between the particles and the wheel loader bucket during granular loading operations, making specialized modeling and simulation techniques necessary. A virtual test environment must include various material types (e.g, rock, sand, dry soil, cohesive soil, wood chips) to accurately represent the wide range of load cases encountered in real-world applications. Thus, an essential engineering need for heavy machinery manufacturers is to develop granular loading simulations that are both efficient and accurate in predicting reaction forces.

A significant challenge lies in balancing speed and accuracy in these simulations.

Although accurate predictions typically require computationally intensive physics-based particle simulations, current industrial practices often rely on overly simplified models that, while faster, lack the necessary precision. This thesis aims to address this gap by integrating AI modeling methodologies trained on data collected from a miniature wheel loader model, with the goal of achieving real-time force predictions with high accuracy.

Beyond force prediction, this research extends to the classification of granular materials. The ability to classify different materials during loading operations has the potential to give self-driving wheel loaders increased awareness of their surroundings. For example, this could help the wheel loader dynamically adjust the magnitude of the forces applied, or loading patterns associated with a specific type of material. More specifically, the machine learning model should explore the potential of identifying differently shaped particles of the same material using sound and force data. In addition, the use of sound and vibrations could complement vision-based material identification methods.

One key advantage of sound-based classification methods is their ability to circumvent issues common in vision-based approaches, such as sensitivity to lighting fluctuations and occlusions caused by weather, dirt, or other external factors [22]. Since loading operations often occur under such variable conditions, sound-based methods could be a superior alternative to vision-based approaches. Previous research has explored audio classification tasks related to loading, such as faulty pouring detection [14] and operational classification [3].

To address the black box problem [10] associated with deep neural networks, which obscures the decision-making process, one or more interpretable glass box models will also be developed and compared. This approach helps build trust in the results by providing transparency. For glass box models, k-nearest neighbors (kNN) classifiers, support vector machines (SVM) [34], and decision trees are commonly used in sound classification tasks. Examples include heart sound classification [38], [34], combined black box and glass box classification tasks [35], and machine state monitoring using sounds and vibrations [12].

1.2 Related Work

In a theses paper, [6] the use of sound data to estimate material mass during pouring tasks is explored. A pouring task refers to the act of loading and unloading material from a bucket repeatedly. This closely relates to the current work, which also involves granular materials and uses a sound-based approach. In addition, the use of poured mass prediction may be applied to the current force prediction case. In the thesis, there are several deep learning architectures, with the long short-term memory (LSTM) and gated recurrent unit (GRU) models proving to be the most effective for processing time-dependent audio data. These models are likely applicable to the current study, as audio data is time dependent. However, the study differs by not incorporating complementary data, such as the exerted forces dur-

ing manipulation, which is a key focus in this thesis. Moreover, while both studies classify distinct particle types, the research does not capture the subtle differences in particle shapes that are central to the current research. Instead, it involves predicting the poured amounts of distinct particles such as rice, pasta, and coffee beans.

In the another paper [27], a robot is taught to predict missing values in a matrix by performing actions like grasping, lifting, and shaking objects. Just as in the current work the study handles the classification of different materials, but with a different output goal. In the study, multiple types of sensor data were used, including robot joint torques and audio. Furthermore, containers with granular materials were shaken and dropped in order to identify the materials in each container. Other than that, additional features such as colors and container weights of each container were used.

Additionally, recurrent architectures that utilize Long Short-Term Memory (LSTM) units have been proven effective in processing acoustic signals. The effectiveness is found in the network's ability to capture sequential relationships [28]. A simpler, but effective approach [4] utilizes GRUs as an alternative to LSTMs for recurrent networks. GRUs are shown to work well in tasks involving acoustic signal learning and have even outperformed LSTMs in certain cases [31, 36]. Furthermore, various studies demonstrate that transforming raw audio into a spectrogram representation is highly effective for training neural networks [11, 29, 25, 26]. As an example, Convolutional Neural Networks (CNNs) achieve notable success when applied to spectrograms for tasks such as speech recognition and other classification challenges [17, 25, 30].

For glass box models, SVMs are successful in handling classification tasks [16]. Particularly, they show over 75% accuracy when "classifying kinds of material and driving stages and styles of Volvo wheel loaders (WLO)". The classification is done in real time, which limits the project to computationally inexpensive machine-learning models. Another study also uses SVMs but for environmental sound classification [34]. The difference compared to the current project is that general environmental sounds are classified and the ESC-10 and ESC-50 datasets are used. These datasets are publicly available and contain mixed audio data e.g., dogs barking, pouring rain, sea waves, babies crying, clocks ticking, people sneezing, helicopters, chainsaws, roosters, and fires crackling. The authors managed to achieve an accuracy of 89% and 65% respectively for the two datasets. A third study focuses on emotion recognition and tests decision trees, logistic regression, support vector machines, and random forest classifiers. The highest accuracy of 76.01% is achieved with logistic regression [1]. A fourth study involves using machine learning to monitor machine health based on sounds and vibrations. The models applied are K-nearest neighbors, convolutional neural networks, and support vector machines. However, the use of synthetic data due to data scarcity likely makes the results less adaptable to real-world data [12].

When handling force prediction, a study [13] employs the Long-Short-Term-

Memory model for predicting the advance rate of the tool during rock boring operations. The study does not include force predictions but uses pressure data as input among other types of sensor data. The data recording and AI-model training aspects of this study are similar to what is carried out in the current thesis. Thus inspiration for research design is taken from the study. Another study focuses on predicting the instantaneous milling forces in real-time based on an input current [23]. In the paper, the nonlinear relationship between the current and output force is highlighted. Therefore, a model utilizing CNN and RNN combined architectures is used. Although this is the case, there is no motivation for why a single RNN would be insufficient.

Next, [18] evaluates an unsteady aerodynamic time series prediction model using an LSTM model. Specifically, the "unsteady aerodynamic force under varied Reynolds number and angles of attack is predicted by the LSTM model." The model is trained on sample intervals from different data sequences and asked to predict the next coefficient in the series. This is highly applicable to the current project. In the end, the model is successful in predicting accurate coefficients and it is observed that when using interpolation techniques for validation it is more accurate than using extrapolation.

Another paper introduces a Neural Network-based method for predicting construction cable forces in large-span concrete-filled steel tubes arch bridges, aiming to enhance both accuracy and efficiency [37]. Both a Multi-Layer Perceptron (MLP) and a Long Short-Term Memory network are fed random cable forces generated by employing a finite element method model of a bridge system. LSTM is deemed the most successful model that achieves the highest accuracy with the least complexity. However, the study utilizes only artificially generated data for training which can lower real-life adaptability.

Finally, the last study [32] aims to model wet milling by creating a computational fluid dynamics-DEM-based framework that combines experiments, simulations, and AI. This framework is used to investigate the impact of disc geometry, tip speed, and grinding bead size in stirred media mills. Once the validity of the simulated models had been confirmed, an AI for predictive modeling of the relative velocity distribution and spatial distribution via heat maps was developed [33]. Regarding the choice of model, genetic reinforcement learning is applied, combining a neural network with a genetic algorithm. The study found that the relative velocities could be predicted with a similar accuracy to that of the simulations. In conclusion, it is found that AI can be used to save computational time compared to simulating each case.

1.3 Research Questions

The aim of the thesis project is to evaluate how machine learning and AI techniques can be used to generate surrogate models for real-time granular material tool interaction simulations. Additionally, it explores the closely related topic of granular material identification.

The main research questions to be answered are

- How do the force predictions made by a machine learning model compare to forces simulated using DEM?
- Which type of machine learning model is best suited for ceramic particle classification based on sound profiles and exerted forces recorded during loading?
- How do interpretable models compare to deep neural networks in both classification and force prediction?

1.4 Research Objectives

The research objectives of this thesis are categorized into primary and secondary goals:

Primary Objectives:

- **Neural Network for Force Prediction:** Develop and train a neural network model to predict forces exerted during interactions between the tool and granular material. This model will be evaluated against simulated DEM force predictions to assess its accuracy and practical applicability.
- **Neural Network for Shape Classification:** Create a deep neural network model capable of classifying three distinct shapes of the same material. The model will be trained utilizing both sound and force data collected during the loading process, with the goal of achieving high accuracy in shape classification.
- **Comparative Analysis of Models:** Develop interpretable models for classification and force prediction, and compare the performances to that of the deep neural networks. This comparison will help determine whether interpretability impacts classification and force prediction accuracy.

Secondary Objectives:

- **Real-Time Data Verification:** Verify the models applying real-time data collected while the robot operates. When real-time verification is not feasible, use the collected data for validation purposes.
- **Model Validation Techniques:** Employ various model validation methods, including cross-validation, and test different end effectors to enhance model robustness. Note, that testing different end effectors may not be possible if real-time operation is not achieved.
- **Sample Collection:** Aim to collect up to 10,000 samples for training purposes. While the goal is to reach this number, the focus is on gathering as many samples as possible to ensure a robust training dataset.

- **Test Other glass box models:** Other interpretable models should be evaluated if possible. There are a few suitable classifiers such as SVM, kNNs and Random Forest.
- **DEM Simulation:** Simulate the bucket-particle interactions in Demify® and compare them to the experimental and predicted force data.

1.5 Limitations

In this thesis, several limitations may impact the results. The dataset consists of 9,999 samples, including data outliers, from complete UR-robot runs, which, while sufficient for training a deep neural network, may still be inadequate for achieving high accuracy. Noise in the sound data could also affect prediction accuracy, even after noise reduction. The choice of model architecture has a significant influence over the produced results. However, only GRU models will be evaluated in this project. LSTMs have been excluded as they were applied to the classification problem initially, but were not able to converge to an accuracy over 5%. Only the three shapes triangle, cylinder, and sphere will be evaluated, excluding other shapes and materials to avoid increased problem complexity. For interpretable models, only one carefully selected glass box model will be evaluated. Additionally, the force prediction model will not be compared against transparent models, and the deep neural network will not be integrated into a real-time system, though it will be ready for future real-time deployment.

2
Theoretical Framework

This chapter presents and discusses the models that are used in the thesis. Among these models are the perceptron, support vector machine (SVM) and the gated recurrent unit (GRU). In addition, DEM is introduced briefly.

2.1 Machine Learning Models

Three machine learning models and their respective mathematical backgrounds are introduced in this section. The three model types are the perceptron, SVM, and GRU networks.

2.1.1 Perceptron

The perceptron machine learning model and is a fundamental part of artificial intelligence [24], [2]. In this thesis, the perceptron is presented without an activation function, which distinguishes it from more advanced deep-learning models. An activation function's main purpose is to introduce non-linearity into the model, allowing it to learn and represent complex patterns in data. The absence of such a function results in a purely linear model, which simplifies its behavior and increases its interpretability. The mathematical representation of a perceptron with weight matrix W, input vector x, bias term b and predicted class $\hat{y}$ is

$$z_k = W_k^T x + b_k \tag{2.1}$$

$$\hat{y} = \arg\max_k(z_k) \tag{2.2}$$

The model computes a weighted sum of inputs, producing scores for each class. The class with the highest score is selected as the predicted class. The model iteratively adjusts weights to improve prediction accuracy using optimization techniques like gradient descent. For force prediction, the output shape can be adjusted to generate either a single force value or a series of values instead of class scores. This modification allows the perceptron to function similarly to a regression model, predicting continuous values instead of discrete categories. Furthermore, the model is characterized by its simplicity, but that also comes with drawbacks. One notable drawback being that the perceptron can only separate linearly separable data [20] making it inappropriate for other types of data. Other than that, the perceptron only provides deterministic outputs that do not indicate the probability of each class being correct [19]. Probabilistic outputs are preferable since they indicate a level of confidence in the results.

2.1.2 Support Vector Machine

The support vector machine (SVM) is a machine learning model which is thoroughly explained in [9]. The method works by separating data using hyperplanes like shown in Figure 2.1 that maximize the margin between different classes. The data points are mapped to a higher-dimensional feature space, where each point represents a feature. Thereafter, the model is trained by solving an optimization problem that minimizes a loss function L subject to certain constraints.

$$L(x, \xi) = \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i \tag{2.3}$$

$$y_i(w^T \Phi(x_i) + b) \geq 1 - \xi_i \tag{2.4}$$

$$3\xi_i \geq 0 \forall i \tag{2.5}$$

x represents the training samples, w the weight vector, ξ is a slack variable which enables a soft margin, y_i the class label for sample i, b the bias term and Φ a non-linear mapping function. The hyper-parameter C controls the trade-off between the complexity and fitting error. This method is referred to as the primary formulation of the problem. However, when the dimensionality of the problem becomes too large, the method might not converge to a desirable accuracy [9]. As a solution to this problem, a dual formulation maximization problem is introduced which utilizes Lagrangian multipliers. Thus, the equations are rewritten

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{2.6}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.7}$$

$$\alpha_i \geq 0 \forall i \tag{2.8}$$

α is a Lagrangian multiplier, K is the kernel representing the shape of the decision boundary and can be chosen either linear or not. Examples of such kernels include

$$\textbf{Linear: } K(x_i, x_j) = x_i^T x_j \tag{2.9}$$

$$\textbf{Polynomial: } K(x_i, x_j) = (x_i^T x_j + c)^d \tag{2.10}$$

$$\textbf{Radial Basis Function (RBF): } K(x_i, x_j) = \exp\left(-\frac{|x_i - x_j|^2}{2\sigma^2}\right) \tag{2.11}$$

$$\textbf{Sigmoid: } K(x_i, x_j) = \tanh(\alpha x_i^T x_j + c) \tag{2.12}$$

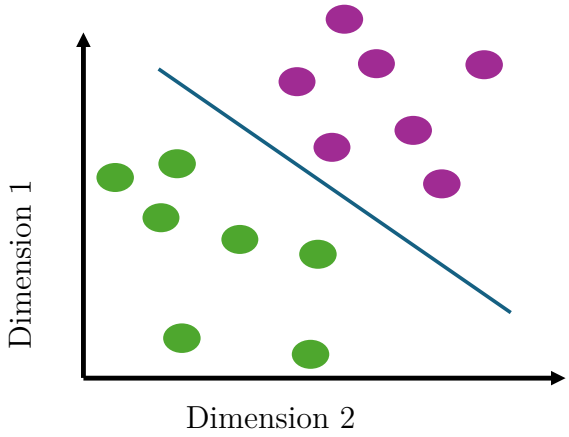


Figure 2.1: SVM hyperplane separating two classes.

Figure 2.1 shows the concept of how an SVM separates data points with the use of hyperplanes. The image displays two classes in a 2D space, which are separated using a linear kernel.

2.1.3 Gated Recurrent Unit (GRU)

A GRU is a type of deep neural network that is described in [8], [5]. The network itself is a form of Recurrent Neural Network (RNN), a model loosely illustrated in Figure 2.2, which typically excels in tasks involving data sequences. Typically, RRNs suffer from the vanishing, or exploding gradient problem. This is a problem for longer data sequences, however, the GRU model reduces this problem significantly. Unlike comparable long-short-term memory (LSTM) networks, which utilize a cell state and three separate model gates. GRUs simplify this process by combining the cell state and hidden state into a single hidden state and using only two gates. Figure 2.3 shows the inner workings of a general GRU. The update gate, z_t, controls how much of the previous hidden state is carried forward. It processes the previous hidden state h_{t-1} and the current input x_t through a sigmoid activation function to produce a value between 0 and 1, where 0 means discarding past information and 1 means keeping all of the past information. $W_{z,r}$ and $U_{z,r}$ represents the weights of the update gate and $b_{z,r}$ is a bias term.

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.13}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.14}$$

The new candidate hidden state $\tilde{h}_t$ is computed using the reset gate, note the use of the $\odot$ Hadamard element-wise product. Once again W_h and U_h are the weight matrices associated with the potential hidden state, g is the hyperbolic tangent activation function, and b_h is the corresponding bias term.

$$\tilde{h}_t = g(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{2.15}$$

Lastly, the hidden state h_t is updated by interpolating between the previous hidden state and the potential state, which was calculated by the update gate:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{2.16}$$

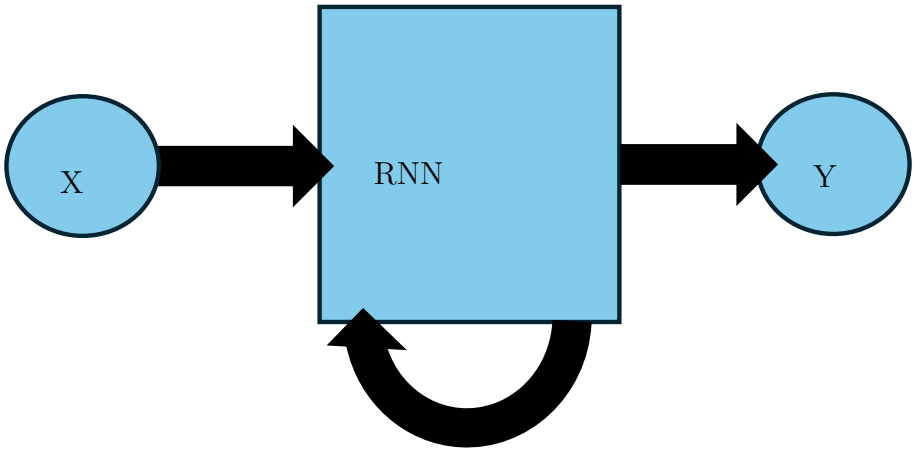


Figure 2.2: A basic RNN architecture.

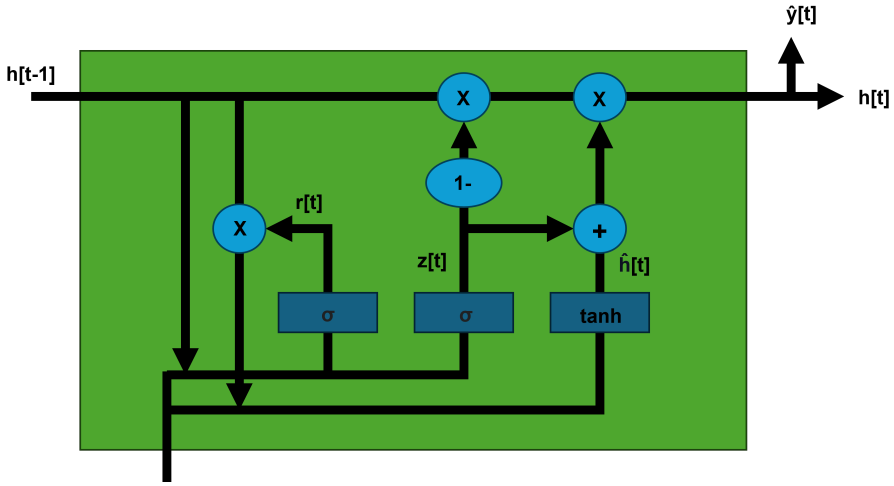


Figure 2.3: A general GRU architecture.

2.2 Discrete Element Method (DEM)

As presented in the objectives chapter, the DEM simulations were compared to experimental data in order to verify the simulation results reliability. Since the simulations are an important tool for evaluation, their underlying mathematical principles should be explained at least on a basic level. Demify® [7], is a high-performance GPU-based DEM software developed in-house at Fraunhofer-Chalmers Centre. The software allows for the simulation of different particle shapes and also integrates with the IPS robotics simulation module. This enables a digital twin setup of the robot loader described in Section 3.5. DEM, or the discrete element method, is best described as a numerical method used to simulate the interactions between particles. Furthermore, each particle is treated as an individual object with

its own unique state variables such as position, rotation and velocity to name a few. These calculations stem from Newton's second law, which more specifically provides the position and velocity of each particle [39].

$$m_i \frac{d\mathbf{v}_i}{dt} = \sum_j \mathbf{F}_{ij}^c + \mathbf{F}_i^g \tag{2.17}$$

$$\mathbf{I}_i \frac{d\omega_i}{dt} = \sum_j \mathbf{M}_{ij} \tag{2.18}$$

In this case, $\mathbf{v}_i$ is the velocity of particle i, ω_i is the rotational velocity of particle i, m_i and $\mathbf{I}_i$ are the mass and moment of inertia of the particle. $\mathbf{F}_i^g$ is the gravitational force applied to particle i. $\mathbf{F}_{ij}^c$ and $\mathbf{M}_{ij}$ are the force and the torque applied to particle i caused by the collision of particle i and j.

Furthermore, it is possible to simulate how particle beds behave under certain circumstances, such as when pressure is applied by an external tool. This makes Demify a useful tool in particle since the granular material that is lifted by e.g., a wheel loader, can be made into a digital twin. This works by considering the collisions between each particle in the particle bed, and it introduces some additional parameters such as particle friction.

Besides the material properties of the particles, the shapes of the particles greatly affect the outcomes of the simulations. In this project, three types of particle shapes are included namely spheres, cylinders, and prisms. The particle shape representation for these particles is handled by using dilated polyhedrons. This method works by sweeping a sphere along the surface of the particle, given that the particle is convex in nature. Formally, this operation is known as the Minkowski sum [21] and its main advantage is its computational efficiency. The operation is shown in Figure 2.4. The broad-phase collision detection in the solver is performed using a boundary-volume hierarchy tree implemented for GPU computation. The contact model used for particle-particle and particle-tool interaction is the Hertz Mindlin-Deresiewicz (HMD) contact model.

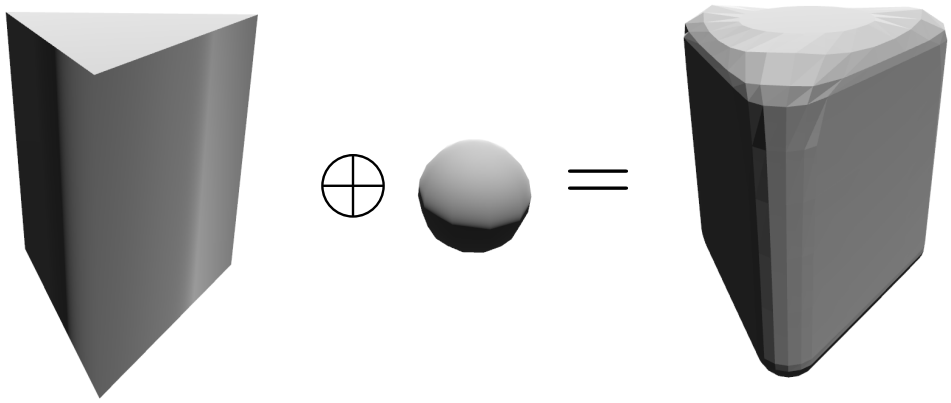
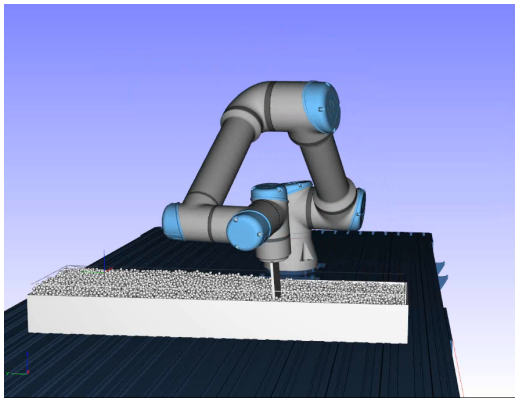


Figure 2.4: The Minkowski sum applied to a prism-shaped particle.

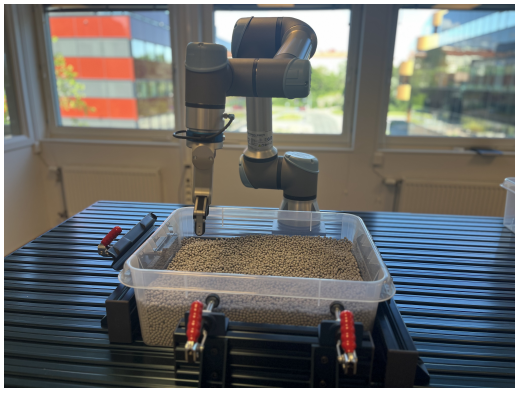
3
Methodology

3.1 Experimental Setup

All training data for the machine learning models is collected using a rig. The rig itself consists of a UR5e robot which has a 3D-printed nylon bucket attached as its end effector as seen in Figure 3.6. Furthermore, the bucket is a down-scaled version of a real-life size bucket provided by Volvo CE. A real-life bucket model is used since the robot is intended to mimic the behavior of an actual loader. Furthermore, using this rig as a wheel loader miniature model is only possible due to the robot's ability to adapt to vastly different situations by choosing the right end effector. Below the robot arm in Figure 3.1 is a box with a horizontally aligned particle bed. The reason for using a flat particle bed is that it is easier to handle mechanically, as e.g, a slope would have to be rebuilt by hand after each operation. Instead, this problem is avoided by introducing a back and forth robot path. With this path, the robot operates in a loop where the particle bed converges to a steady state after a certain number of continuous operations. The particle bed itself contains ceramic tumbling media of the shapes cylinders (F 6x7 Z, art-nr 1205803), prisms (BA 6x6, art-nr 1205870) and spheres (FSG Ø 6, art-nr 1205784). These have all been measured to fill approximately the same volume in the box. Figure 3.1a) shows an example of a digital twin setup in Demify®.



(a) Demify® particle simulation with UR-robot.



(b) The UR5e robot rig.

Figure 3.1: Real-life UR5e robot (b) and its digital twin (a).

Although several types of data are recorded from the robot setup, the two most important data types are the forces and the recorded audio. The forces are recorded from built-in sensors in the UR5e robot, while the audio is recorded using an external microphone. The microphone is a Superlux ECM999 condenser microphone which is attached to a mic stand and connected to the external Focusrite Scarlett 2i2 3rd Gen audio interface. The audio interface is then connected directly to a laptop which is used to control the robot movements and data collection. The scripts for data collection were created based on the universal robots real-time data exchange (RTDE), which is an external python package. RTDE's enables for easy data logging between the universal robot and a computer. Normally, the UR-robot lacks the functionality to record data in real-time, or at a certain frequency. This functionality is what is gained from using the RTDE package, although the maximum sampling frequency is 500 Hz for the universal robots e-series. The actual data collection script is based on a universal robots RTDE setup script, which is then modified to handle both the audio and the robot data collection simultaneously. The data collection syncing is handled by letting a function monitor both the force and audio recording functions, making sure that the functions wait for each other in the case that one finishes before the other.

3.2 Dataset

In order to train the selected black and white-box models, an appropriate dataset should be used. Normally, open database datasets such as MNIST, ESC-50, Urban-Sound8K, Kaggle and others would be valid candidates for such tasks. However, in this project a specialized dataset with the classes prisms, spheres and cylinders was created to mimic the conditions during normal excavation. A simple setup using a UR5e robot and three containers with one class in each was used. The volume in each box was held as consistent as possible by taking into account the packing and material densities of each particle class. However, the prism box had to be filled with a lower volume as the robot failed to run through the material due to load limits. The path of each operation was kept consistent, but the speed was varied for each sample as to broaden the applicability of the dataset. More precisely, the mean speed was set to 50% of the robot's maximum tool speed of 1 m/s and varied according to a normal distribution with a standard deviation of 10%.

The dataset consists of 9999 samples with 3333 samples belonging to each class. The collection of the data took approximately two weeks to complete with around six hours of running time each day. Furthermore, each sample consists of a sound recording and a corresponding force data recording with a duration of approximately 16 s for each sample. Additionally, the sampling rate of the sound data is 48 kHz while the force data was recorded at 200 Hz. The difference in sampling rate could potentially introduce syncing issues between the audio and force data. However, this is mitigated through the use of global time and syncing tests. In the syncing tests, impulse-like sound signals are generated as a way to sync max sound impulses to max forces.

One problem with the data collection is that syncing between the audio data and the robot data is difficult to ensure. In order to combat this problem, a simple syncing test is performed as shown in Figure 3.2. The test has the robot come in contact with a wooden plank, holding down on it with a force of 30 N for four seconds before release. Since the sound with the highest amplitude appears at the same time as the highest force, the signals are considered to be in sync. If they are not in sync this results in the machine learning models learning relationships between data that are incorrect.

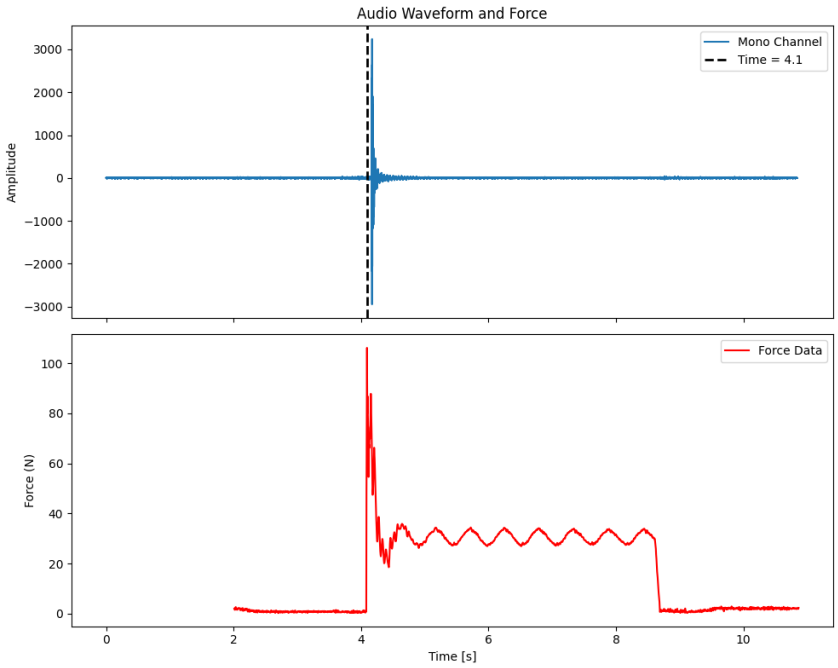


Figure 3.2: Calibration test applied force: 30 N.

3.3 Data Augmentation Techniques

Data augmentation is an important tool when training robust and fast-converging machine learning models. Especially helpful is the use of data normalization which can increase the convergence rate during training. Data normalization does so by centering the data, often around zero mean with a standard deviation of 1, which makes it easier for the model to learn patterns and relationships within the data more consistently. Different types of normalization are often applied to a select number of samples, rather than the whole dataset. However, normalization is an exception because of its great benefits. Therefore, it is applied to the full dataset.

In the case of the audio-related features the Mel-Frequency Cepstral Coefficients (MFCCs) are applied. The MFCCs are features commonly used in speech and audio processing to represent the power spectrum of a sound signal. They are derived from the cepstral representation of the audio signal, with the frequency axis scaled according to the mel scale, which approximates how the human ear perceives sound

frequencies. MFCC itself most often includes 13 unique features.

Zero padding is used to make the input size of each sample equal, which is a requirement when training most machine learning models. In this case, zeros are added to the end of all data samples shorter than the longest identified sample. Although all samples must be padded to the same length, zero padding is only applied to data that is cut into windows for the GRU architecture. This is because, when training the SVM and perceptron the full image is treated instead of window snippets. In the single image, no matter its length only a single instance of each feature is extracted, making the original image length irrelevant.

Other data augmentation techniques are applied to a fraction of the samples. These techniques aim to make the model more robust to diverse sample types. Essentially, augmentation is meant to artificially expand the dataset and make it more diverse by adding changes to some of the data samples. However, one drawback to this approach is that augmentation cannot replace the use of new fresh samples. Instead, augmented data only extend the adaptability of the dataset within the boundaries of what is already known, without introducing truly novel information. Applied augmentations are Gaussian noise, data reversing (flipping), time shifting, time masking, pitch shifting and scaling.

- **Gaussian Noise**: Adds random noise to simulate background conditions, like those found at a construction site.
- **Data Reversing**: Flips the data, playing it in reverse, which introduces temporal variability.
- **Time Shifting**: Moves the signal along the time axis, altering the timing of events in the sample.
- **Time Masking**: Temporarily hides parts of the signal, forcing the model to make predictions with less information.
- **Pitch Shifting**: Changes the pitch of the audio data, making the tones higher or lower to simulate variation in frequency.
- **Scaling**: Increase or decrease the data amplitude

3.4 Model Architectures

Many of the machine learning models used in this project are very similar. The main applied models are GRUs, SVMs and Perceptrons with some variations. This section aims to clarify what configurations are used for these models as well as type choice of input data to the models. After all, more than the raw data itself can be fed into a model, specifically data features. This can be seen as giving the model directions for what to look for in the data.

3.4.1 Deep Learning Model for Particle Classification

The chosen deep learning models used for particle classification is a GRU model since it can handle time series data as well as mitigate the vanishing gradient problem.

The inputs to the network consist of

- 1-13: MFCC features
- 14: mean force
- 15: median force
- 16: force variance
- 17: max force
- 18: max force time

The GRU network architecture shown in Figure 3.3 consists of a single GRU layer that takes both the 18 available input features as well as an initialized hidden state h0. The value 2097 in the figure refers to the sequence length of the data. After the GRU layer, a layer normalization is applied, which normalizes the data across the feature dimension. Layer normalization is preferred over the common batch normalization when working with RNN-based networks which is because batch normalization may introduce instabilities since time-dependent data varies between batches. The normalization itself is beneficial since it can accelerate the training process. After normalization, the data is output through a fully connected layer which reshapes the data into a suitable 1D output format. Lastly, a softmax function calculates the probability of each outputted class.

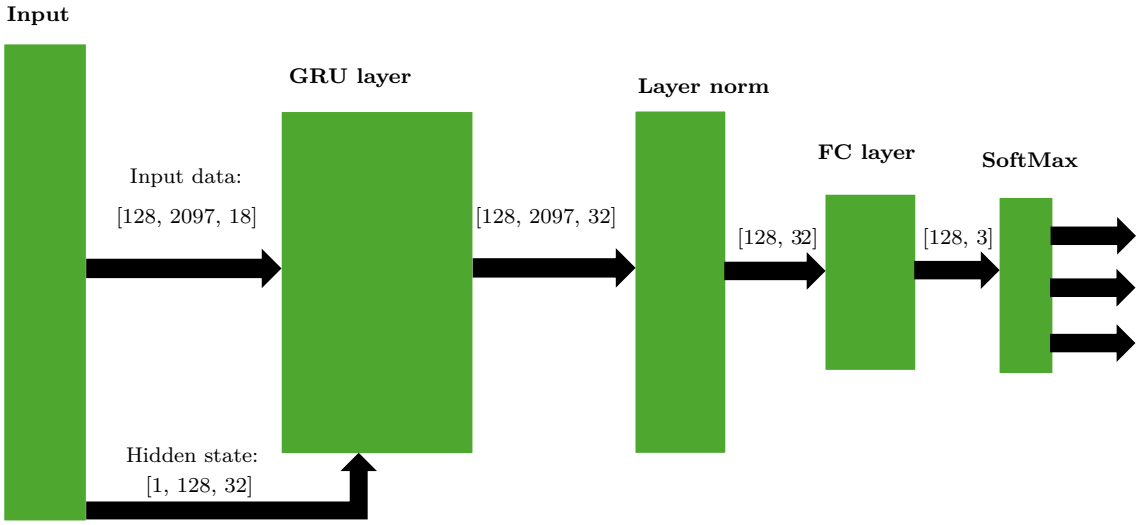


Figure 3.3: GRU network architecture.

3.4.2 Interpretable Model for Particle Classification

Although there are many suitable choices for interpretable machine learning models, it is good practice to start by using simple models. In this case, a simple perceptron was selected as an initial architecture. Although perceptrons usually only handle binary classification tasks, they can be adjusted to handle multiple classes with little difficulty. This is done by adding a new row of weights for each available class, which results in a weight matrix of the shape classes $\times$ (features + 1) with the additional +1 representing a bias term. Furthermore, the perceptron takes the same parameters as the deep learning model, namely:

- 1-13: MFCC features
- 14: mean force
- 15: median force
- 16: force variance
- 17: max force
- 18: max force time

Figure 3.4 shows the architecture of the perceptron. n inputs are fed to the perceptron, which then applies weights that are summarized into three votes. There is a single vote for each available particle class, and the class with the highest score is output as the final class.

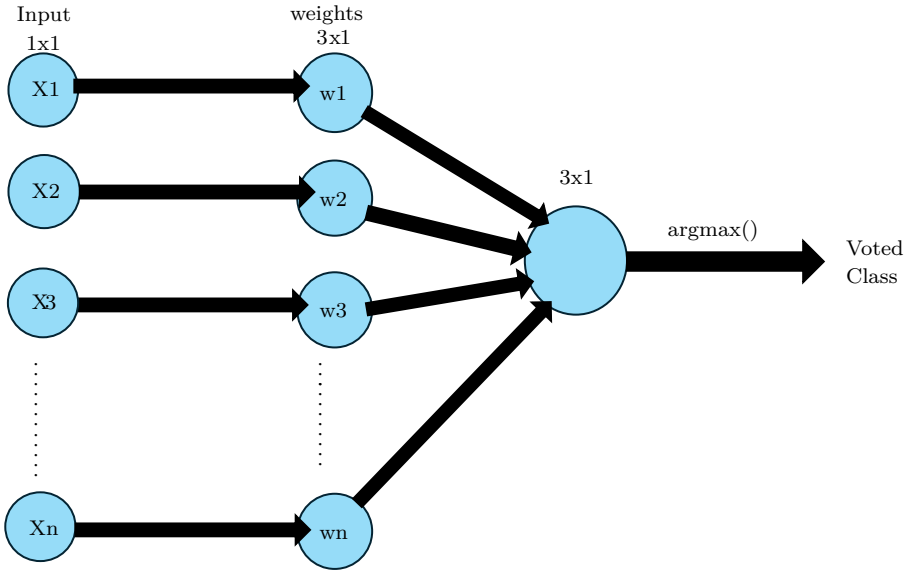


Figure 3.4: Perceptron architecture.

Other than the perceptron, an SVM model is trained to investigate if the use of kernels with different complexities will produce better results than the linear perceptron model. The input data is kept in the same format as for the other classification models to preserve a level of consistency, which is important for the final evaluation. Note that the SVM can be considered both interpretable and not depending on the choice of kernel, where a linear kernel results in linear weights comparable to the weights of a perceptron. On the other hand, an RBF kernel introduces a higher complexity which can not be considered a "glass box" model. However, for comparison simplicity all of the SVM results are included in the same subsection.

Although many other models, such as decision trees and kNNs are suitable, the SVM model is chosen, because it can handle non-linear and high-dimensional data. Furthermore, unlike kNNs, which perform minimal work during training but must process the entire dataset at each evaluation, an SVM performs heavy computations during the training phase. Thus, it may be more time-efficient when making predictions post-training. Thus, an SVM is more suitable for larger datasets, despite the fact that 10,000 samples in this case result in a long training time. An additional advantage of SVMs is that they are less prone to over-fitting. The SVM is trained

using a 5-fold cross-validation technique. The technique splits the dataset into five subsets and one of them is used as the test set while the others are used for training. This procedure is repeated with different folds as the test set each time. This makes the model less dependent on a specific train-test split, which reduces the risk of over-fitting.

3.4.3 Deep Learning Model for Force Prediction

The step from training a model to generating full-force sequences can be confusing. One could argue that it is enough to use a simple regression model, or something very close to a curve-fitting model. However, in this case, the optimal scenario would be to generate forces similarly to how they are generated with DEM. One way to do so is by applying a one-step approach, where only the next force value in a sequence is predicted. This is meant to mimic the dynamic one-step calculations of DEM, where in a real-time system positions could change stochastically. Therefore, because of the one-step force generation, two steps are required for the full force to be generated. The first step is training the actual model on some data and the second step is creating a feedback loop, which helps the model generate a force graph dynamically.

Choosing the right model when trying to perform a force prediction is crucial to the success of the task itself. Without a fitting approach, the model may not be able to learn the underlying patterns in the data and make successful predictions. Like in particle classification, a GRU was chosen as the base network for the force generation task. However, not only does the type of model dictate how well it will perform, but the format and the type of data that is fed to the network will greatly affect the outcome as well. With this in mind, a total of four data formatting approaches are tested.

The first one is a window-based approach, where the data is divided into a number of windows before training. At the stage of training one random window is selected from the current sample and fed as an input to the network. The network then tries to predict the first force value of the next consecutive window. The advantage of this approach is that when selecting a random window it should be possible for the model to make correct predictions no matter when it starts making predictions during the material loading process. This should result in a model that is better at generalizing than one that requires all previously known data in a time series. The data types that are fed to the network are positions, velocities, accelerations, and previous forces in the selected window.

The second approach, which is henceforth referred to as the gaze-forward approach, is very similar to the first approach. However, instead of only making a single forward prediction, the model must predict a larger part of the time series. The reasoning behind this method is that the model becomes better at predicting the next value in a time series when it is trained to predict more than only a single value. Although a greater part of the force is predicted, the feedback loop should

still only use the first predicted value.

The third method moves away from the use of windows and lets the network use all previously known data points as input. This means that the model might not be able to handle partial input sequences as well as the others. However, one advantage of this method is its simplicity. The use of windows introduces complexity, which could also result in unexpected model behaviors. In addition to that, the model takes the derivative of the force as an additional input, giving it another hint to what should be the next force value.

Finally, a simple model similar to the previously mentioned windowless model is trained. The difference compared to the other model is that it predicts the full sequence compared to just a single value. In addition, it utilizes a different set of input data. In this case the input is position, velocity, acceleration, and particle type. Note that the force is no longer used as an input to the network, instead its is only required as an output from the model.

3.4.4 Interpretable Model for Force Prediction

The interpretable models in the force prediction case are the same as in the classification case. The interpretable models utilized are once again a perceptron, or regression model as well as an SVR, which is an SVM that has been adapted to handle time-series data instead of classification. Compared to the classification case, the prediction models take other inputs. In addition, the perceptron model is not averaging like the implemented particle classification perceptron was as it was far easier to set up a regression model based on a neural network architecture rather than creating one from scratch. This was largely a decision made due to time constraints, but it should have a low impact on the final results.

3.5 DEM Modeling

A software extension called Demify®, designed for particle interaction simulations, serves as a valuable tool for verification purposes. As mentioned, one of the main goals of this project is to create a force prediction model that can predict forces in the same way Demify® does but at a faster rate, or preferably real-time. As it stands, the setup in Figure 3.5 needs up to three hours for a single force prediction. The digital setup itself consists of a box, particle beds with the three particle shapes, and a bucket. All these components were created with the same dimensions as their real-life versions. The bucket is identical to the one attached to the robot arm and the box has the same volume as the box in the rig. However, the box itself is a simplification of the real box which has more details and a slightly irregular shape as seen in Figure 3.1b. Figure 3.5 shows the simulation setup in Demify®. The bucket path is displayed in red, blue, and green which corresponds to the x, y, and z coordinates of the bucket at each time step. Furthermore, the time steps for this operation are kept the same as in the real robot rig with a frequency of 200 Hz.

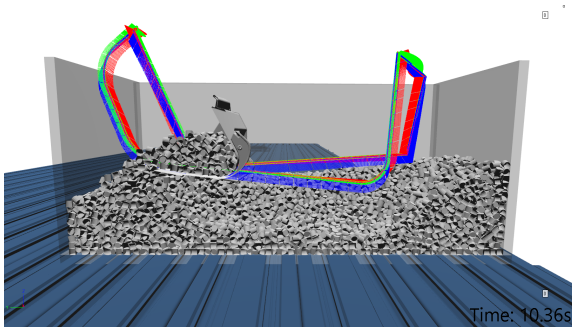


Figure 3.5: Demify® Simulation Setup.

As for the choice of parameters related to the particle-particle (pp) and particle-tool (pt) interactions a few simple tests were conducted to find the friction and restitution. The mean friction of the particles, which are made of the same material, is around 0.38. The coefficient of restitution is around 0.48 for the ceramic particles. A parameter study was also conducted in Demify® which evaluated all pp and pt combinations between values 0.3-0.8 in steps of 0.05.

The bucket has to follow the same path as the UR-robot does. This was ensured by exporting the positions from the robot itself and converting them to an xmo-file containing the path in a quaternion-based format. Figure 3.6 shows the CAD version of the real-life bucket, which was used to 3D-print the bucket itself. In addition, the CAD model was imported into Demify® as the bucket object seen in Figure 3.5

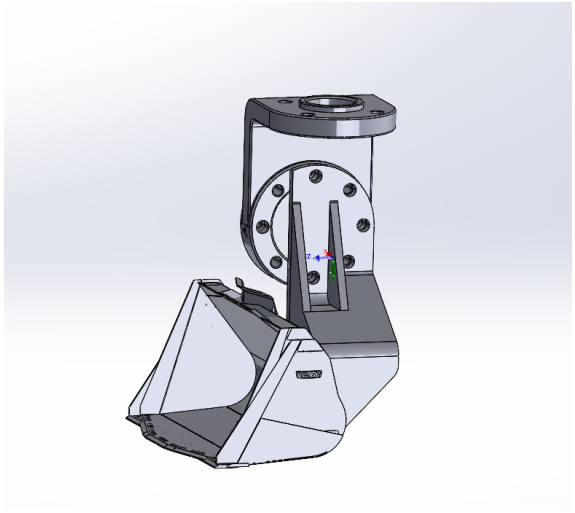


Figure 3.6: A CAD model of the down-scaled bucket and its UR-robot attachment.

3.6 Training Process

The training process is determined by the way the data is handled, how the data is loaded, and by the choice of parameters. The parameters shown in this section are the parameters that produced the best results for each model. However, many different combinations have been tested in this project.

3.6.1 Training of the Classifiers

The training of the GRU network is done using PyTorch, and the training pipeline starts by loading the data through a custom data extraction class. Instead of loading all data at once, it is loaded dynamically, which means that data is only loaded right as it is being used. Furthermore, the custom class both augments and extracts data features in that order. In this case, the temporal aspects of the data are especially important and therefore each sample is divided into sub-windows so that the temporal aspects are not lost. Secondly, the data is loaded into a DataLoader, which is a commonly used package that loads the data into batches and shuffles it. The use of batches accelerates model learning as the model is allowed to update its weights after every batch instead of having to go through the entire dataset to take a step. The batch size is chosen to be 128 samples per batch. Other than that, the window length is 2097 for the GRU network. The input_size is the number of features fed to the network, the hidden_size is the number of neurons in the hidden state vector h_t. num_layers is how many GRUs are stacked on each other, the learning_rate is how much the weights are updated at each time step. The num_epochs are the total number of times the whole dataset is processed by the model during training and the sequence_length is the length of the data loaded into the network. After initializing the data loader, the hyper-parameters are tuned. Note that input_size is the same as the number of features. The hyper-parameters are

- input_size = 18
- hidden_size = 32
- num_layers = 1
- output_size = 3
- learning_rate = 0.001
- num_epochs = 20
- sequence_length = 2097

When the hyper-parameters have been chosen the data is loaded batch by batch for training. Each batch is then processed by the model until the whole dataset has been processed, which corresponds to finishing one epoch. When the specified number of epochs has been run, the model is fully trained. The algorithm used for the GRU model is presented in Algorithm 1.

Algorithm 1 Training a GRU model.

Inputs: X (training data), y (target labels), num_epochs, $batch\_size$, $learning\_rate$
X, y shape $(num\_sequences, num\_features, num\_samples)$
Outputs: Trained GRU model
$model \leftarrow$ initialize GRU model
$optimizer \leftarrow$ initialize optimizer (Adam)
$loss\_function \leftarrow$ initialize loss function (CrossEntropyLoss)
for epoch = 1 to num_epochs **do**
    for batch_X, batch_y in dataset **do**
        $output \leftarrow model(batch\_X)$ ▷ Forward Pass
        $loss \leftarrow loss\_function(output, batch\_y)$ ▷ Compute Loss
        optimizer.zero_grad() ▷ Reset Gradients
        loss.backward() ▷ Backward Pass
        optimizer.step() ▷ Optimize Model Parameters
    print("Epoch:", epoch, "Loss:", loss) ▷ Log training progress
Return trained GRU model

The Perceptron model is trained on non-sequence data and thus it has one less dimension compared to the deep learning models. In this case each sample is treated as an image rather than a sequence of windows and n features are calculated per sample instead of $n \times num\_windows$. When training the perceptron model the entire dataset is loaded and kept in memory before the training phase. The hyper-parameters are:

- num_classes=3
- num_features=18
- max_iter=500

num_classes refers to the number of classes that the classifier can classify the data into. num_features is the number of extracted features and max_iter is the number of allowed training epochs before stopping. The model architecture represented in pseudo-code is

Input variables

N – number of samples
d – number of features
X – training data [N, d]
y – true class label [N, 1]
$num\_classes$ – number of classes
η – learning rate
max_iter – max number of epochs

Algorithm 2 Multiclass Averaging Perceptron.

$X \leftarrow$ add a bias column of 1's to X
$W \leftarrow$ initialize weights as zeros of shape [num_classes, d+1]
$W_c \leftarrow$ initialize cumulative weights as zeros of the same shape as W
$total\_updates \leftarrow 0$
repeat max_iter *times*
for i = 0 to N - 1 **do**
    $x_i \leftarrow$ input vector for the i-th sample
    $y_i \leftarrow$ true class label for the i-th sample
    $scores \leftarrow W \cdot x_i$
    $\hat{y}_i \leftarrow \text{argmax}(scores)$
    if $\hat{y}_i \neq y_i$ **then**
        Update weights:
        $W_{y_i} \leftarrow W_{y_i} + \eta \cdot x_i$
        $W_{\hat{y}_i} \leftarrow W_{\hat{y}_i} - \eta \cdot x_i$
        Update cumulative weights:
        $W_c \leftarrow W_c + W$
        $total\_updates \leftarrow total\_updates + 1$
if $total\_updates > 0$ **then**
    $W \leftarrow \frac{W_c}{total\_updates}$

Lastly, the SVM model can be trained both with or without windowing the data. In either case, the data is loaded just like in the perceptron case, or loaded fully into memory before training. In this case, the validation data are not used since the built-in scikit SVM function does not have an option to include it. As just mentioned, a built-in model function is used for training. The function is especially made for support vector machine classification and a few hyper-parameters are required and an example of such a configuration is

- kernel = "poly"
- C = 1
- γ = "auto"

The kernel parameter decides which type of data divisor should be used. "Linear classifies data utilizing dividing linear planes while "poly" utilizes polynomials for data division. C controls the trade-off between model complexity and over-fitting. γ decides the curvature of the decision boundary and 1/(num_features$\times$ X.var()) is the value which γ is set to when auto is applied. There are many different types of kernels that can be used to divide the feature space, but in this case the main three investigated types are poly, RBF and linear kernels. Furthermore, varying these three kernels together with the windows/no windows variable a total of six models are trained as variations of the SVM. These models being

- no windows - linear kernel
- no windows - polynomial kernel
- no windows - RBF kernel
- windows - linear kernel
- windows - polynomial kernel
- windows - RBF kernel

Algorithm 3 support vector machine.

$X \leftarrow$ input feature matrix
$y \leftarrow$ true class labels
$X \leftarrow$ scale features (optional)
model $\leftarrow$ SVC(kernel='rbf', C, gamma) (initialize the model)
Training Phase
for iteration = 1 to *max_iter* **do**
    formulate the dual optimization problem
    use Lagrange multipliers to maximize margin
    apply kernel trick for non-linear data
    $W \leftarrow$ update weights based on support vectors
$model.fit(X, y)$ (train the model)
Prediction Phase
$X_{\text{test}} \leftarrow$ input test feature matrix
predictions $\leftarrow$ $model.predict(X_{\text{test}})$ (make predictions)
Evaluation
accuracy $\leftarrow$ evaluate accuracy with true labels
print("Model Accuracy:", accuracy)

3.6.2 Training of the Force Prediction models

The force prediction models are trained in almost the same way as the classifier models but with different hyper-parameters and some tweaks to the data augmentation. The first window-based model referred to in Section 3.4.3 has 7121 weights in the network and an allowed error of 5%. The included features are positions, velocities, accelerations, window forces, the window force derivative, and the last force in the window. The hyper-parameters are:

- input_size = 18
- hidden_size = 40
- num_layers = 1
- output_size = 1
- learning_rate = 0.001
- num_epochs = 14
- sequence_length = 250

The second model predicted a full window of length 250, and the total length of each sample is 5247 data points. In addition, the batch size is set to 128. The model has 17210 weights in total which results in a large model. Furthermore, the included features are the positions, velocities, and accelerations of the tool along with the forces in the chosen window. The last force in the window is also included as a feature to emphasize the importance of the most recent value compared to other values in the time series. The hyper-parameters are:

- input_size = 17
- hidden_size = 40
- num_layers = 1
- output_size = 250
- learning_rate = 0.001
- num_epochs = 25
- sequence_length = 250

The third model, which does not include windows, has a batch size of 128, 7001 weights. The model has 17 features which are the tool positions, velocities, accelerations, the force data, and the derivative of the force data. and the following hyper-parameters:

- input_size = 17
- hidden_size = 40
- num_layers = 1
- output_size = 1
- learning_rate = 0.001
- num_epochs = 9
- sequence_length = 5247

The fourth model has a total of 221967 weights and only considers the position, velocity, acceleration, and particle type during training. These result in a total of 16 features.

- input_size = 16
- hidden_size = 40
- num_layers = 1
- output_size = 5247

- learning_rate = 0.001
- num_epochs = 20
- sequence_length = 5247

Lastly, the modified perceptron, also referred to as a regression model, has the same parameters as the fourth GRU model.

3.7 Evaluation Metrics

All the model performances are measured using various methods. As expected, the test accuracy is computed to see how each model performs on the test dataset. In addition, the training and validation accuracies are computed for each epoch during training, which makes over-fitting more easily detectable for all models except SVM which does not allow for validation data. Over-fitted models often have a significantly higher training accuracy compared to validation accuracy. This is because the model is good at predicting based on the known training data, but is lacking when trying to classify unseen data. Other than that, a confusion matrix is computed which shows which classes are more likely to be confused with each other. In the force prediction case the accuracy cannot be computed as easily as in the classification case. In this case, a margin of error is selected and the models is deemed successful in its prediction if the predicted value falls within the selected margin. The margin is chosen to be 5% for all prediction models. Note that the choice of allowed error does not affect the model itself during training but it affects the training accuracy convergence. Other than that, the experimental data is used for evaluation and is compared to both the simulated and AI-generated forces in the force prediction case.

3.8 Ethical Considerations

The ethical implications of the project should also be taken into consideration. One observation is that projects involving data should be as transparent as possible with how the data is recorded and what it is used for. This ensures people's privacy and although no one is recorded during the collecting of samples, other problems could arise if someone else decided take inspiration from this thesis. For example, a company might decide to save the sound data while an excavator with an attached microphone is running. Such excavators could be put into public spaces and record humans without their knowledge. This could be considered a breach in privacy and it might lower people's perceived freedom. Also, biases in data should be clearly presented as to not purposely skew any interpretations. Other than these ethical aspects, it is also important that the sampling equipment does not harm people or animals. While collecting this data it is important that it is done in a safe and supervised manner. Thankfully, I have undergone a course in how to run the robot safely and a safety area should be set up around the robot before collecting data.

4

Results

This chapter presents the results of the trained models. First, the particle classification models are presented by showing their learning curves and their tendency to confuse particles during classification tasks. Next, the force prediction model results are shown and these consist of two window-based GRU models, one non-window-based GRU model but with force derivatives as input, the other GRU model with positions and particle types as input, and a regression model with the same input as the previous model. Lastly, the simulation data generated with the Demify® software is compared to both the AI-generated data and the experimental data. This highlights the models' strengths and weaknesses in replicating empirical force behaviors.

4.1 Particle Classification

In the following section, the results from the black box and glass box classification models are presented.

4.1.1 Deep Learning Model

The first model to be evaluated was the GRU meant to classify the three particle shapes based on sound and force data. The training and validation accuracy convergence appear in Figure 4.1 where the percentage of correctly guessed particle types is displayed on the y-axis. The y-axis displays the number of epochs, where one epoch means that the model has processed and made predictions on the whole dataset one time. As can be observed in the figure, the model starts to converge after around ten epochs. Furthermore, the validation and training accuracies increase similarly, which does not point to any direct instabilities or over-fitting. The model managed to converge to a final training and validation accuracy of approximately 91%. The final test accuracy reached 91% as well, but is not shown in the plot. In conclusion, this model shows promise and is mostly successful in predicting particle types. The training time was around three hours when run on an RTX 3060 Ti.

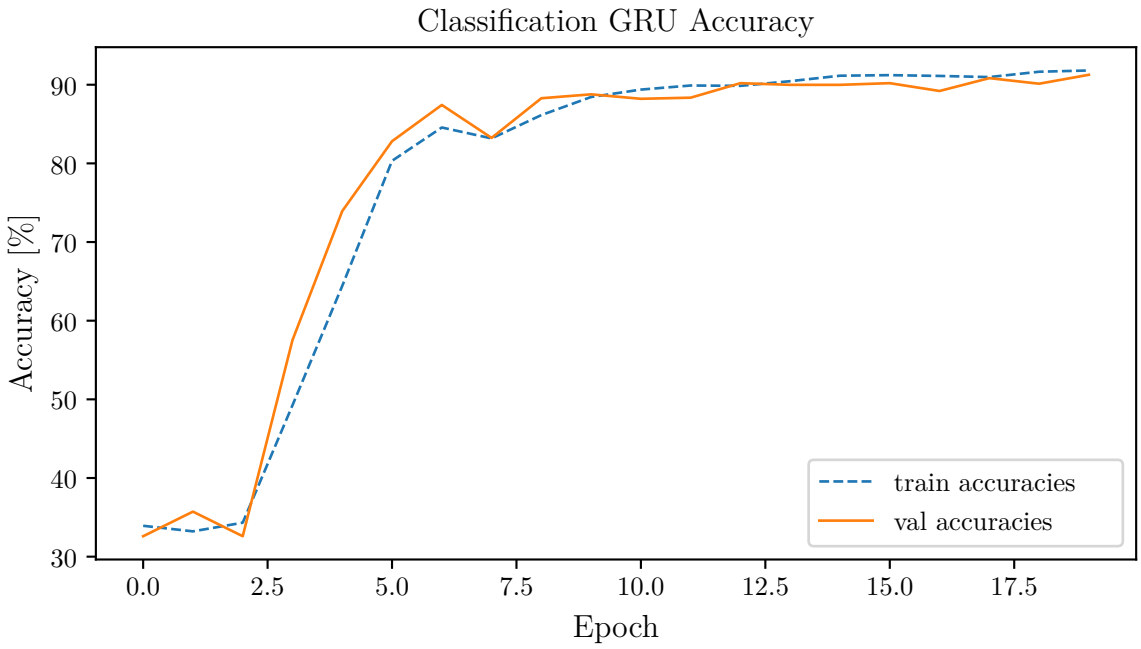


Figure 4.1: Classification GRU accuracy during training.

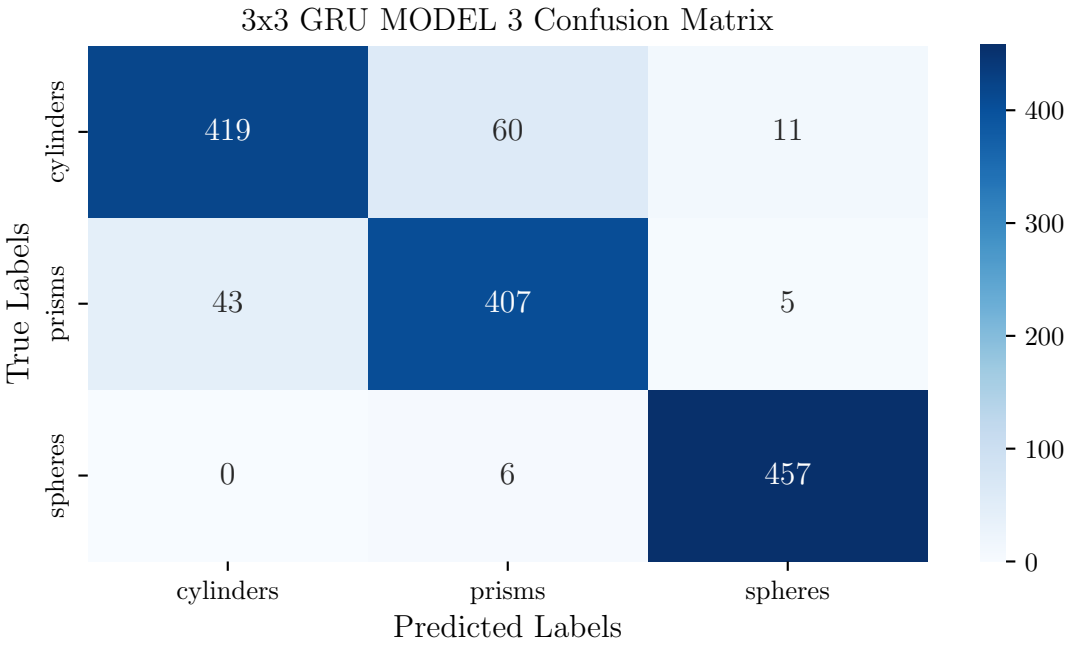


Figure 4.2: Classification GRU confusion matrix.

The matrix in Figure 4.2 shows the three possible classes, namely cylinders, prisms and spheres and how the model confuses the types with each other during particle type classifications. The x-axis represents the particle type predicted by the model, while the y-axis represents the actual particle types. Each value in the matrix corresponds to a single prediction and high values on the diagonal implies that the model is able to make accurate predictions. Furthermore, the confusion matrix itself is generated based on the test dataset, which does not share data with

either the training or the validation dataset. In this particular case, the values appear to be high on the diagonal compared to other positions. The greatest confusion appears between the prism and the sphere shape as 59 samples were incorrectly classified as spheres instead of prisms. A similar confusion occurs the other way around as well. In conclusion, this confusion matrix indicates a low confusion between the particle types. This helps in validating the GRU prediction model and shows that the model itself is successful in making correct predictions based on sound and force data, even when the data is unseen.

4.1.2 Interpretable Model

Figure 4.3 illustrates the training and validation accuracies of the perceptron model designed for particle classification. The x-axis shows the epoch in training, while the y-axis displays the achieved accuracy at each epoch. It is observed that the training accuracy converges to around 87% while the validation accuracy achieves a greater final accuracy of 99%. This difference in convergence is likely due to the augmentations applied to the training dataset. The training dataset was modified to be more generalizing than the original data, but the validation and training datasets were only normalized. Consequently, it should be easier for the model to classify the particles in the validation set since the data lacks distortions. The perceptron model itself managed to converge after approximately 500 epochs, which required a training time under 30 min. After training, the model was tested on an independent test dataset, which was classified with an accuracy of 99.53%.

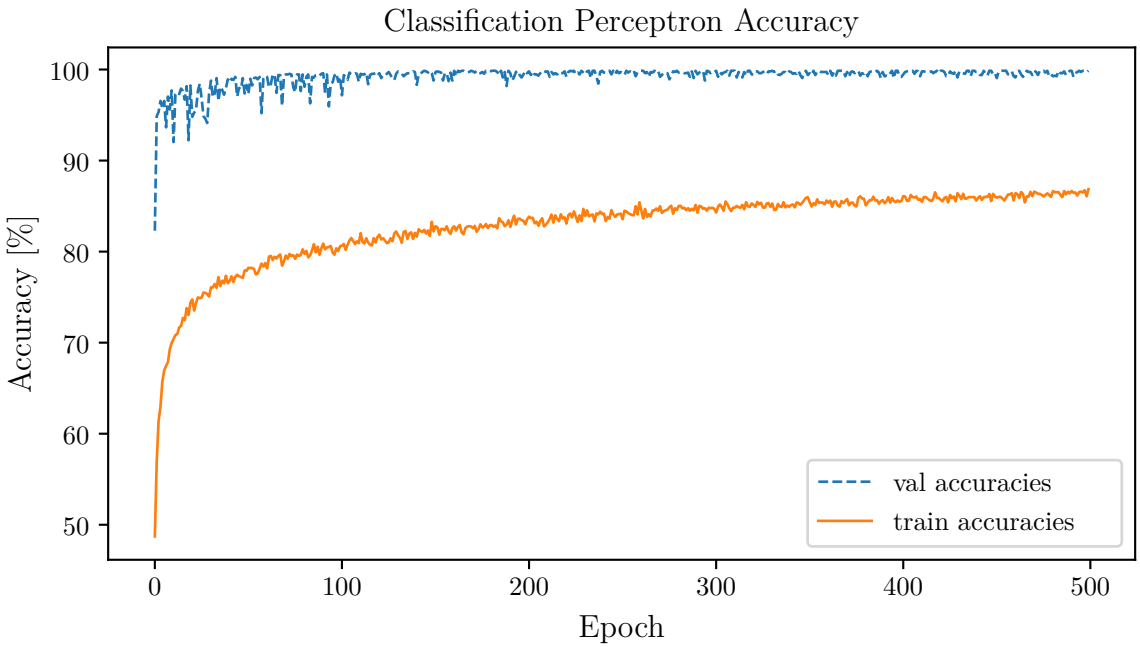


Figure 4.3: Training and validation accuracy during perceptron training.

In Figure 4.4 a heat map of the perceptron model parameters is shown. This heat map shows the classes, 0-cylinders, 1-prisms, 2-spheres, on the y-axis. The x-axis displays each of the features given to the model during training. As for the interpretation, the light to dark color scale represents how large the impact of a certain feature is on each class. Furthermore, the feature with the largest impact for each particle type is circled in blue, e.g, feature 3 contributes the most to the correct classification of the spheres. As mentioned previously, the first 13 features are MFCC and related to audio, while the remaining features include information about the forces.

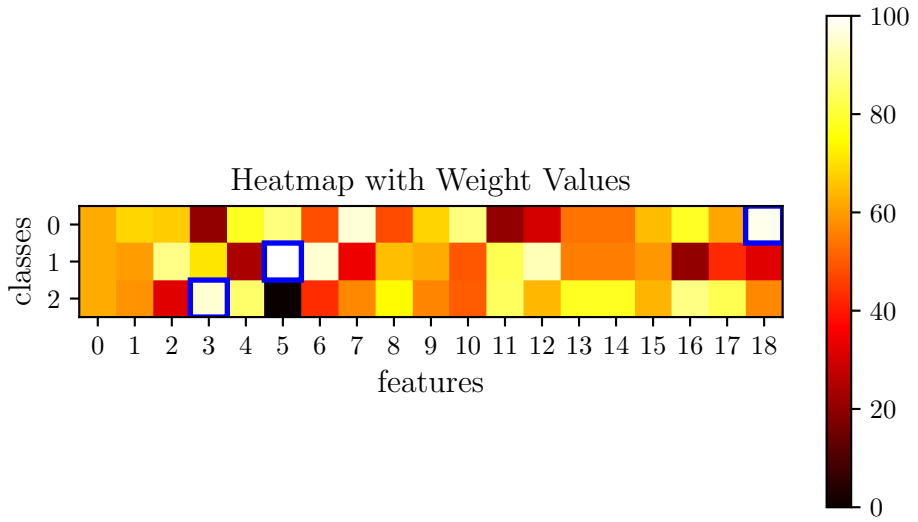


Figure 4.4: Perceptron Heat Map.

The confusion of the perceptron model appears in Figure 4.5 and since this is also evaluated on the test set, it was observed that the model achieved a high accuracy just like indicated in Figure 4.3. This is shown by the strong diagonal of the confusion matrix. The SVM model was trained with three different kernel types, Linear, Polynomial or a Radial Basis Function (RBF), as well as with windowing or not. This resulted in six different variations of the SVM model. Since the SVM lacks the same training functionality as a neural network, an epochs-based training accuracy could not be utilized. In Table 4.1 the final results from the SVM training are displayed.

Table 4.1 includes the achieved accuracies when feeding the training dataset to the SVM. As an example a linear kernel that utilizes windowed input data achieved an accuracy of 96%. The cases where no windows were used produced better results, as the model was able to achieve a perfect classification score no matter which kernel was chosen. Furthermore, the optimal kernel and input format according to the test accuracy is the exclusion of windows and with no preference towards a kernel type. Next, Table 4.1 also contains the training accuracies for all case combinations, and it is observed that the accuracy achieved from the training set is slightly lower compared to that of the test set.

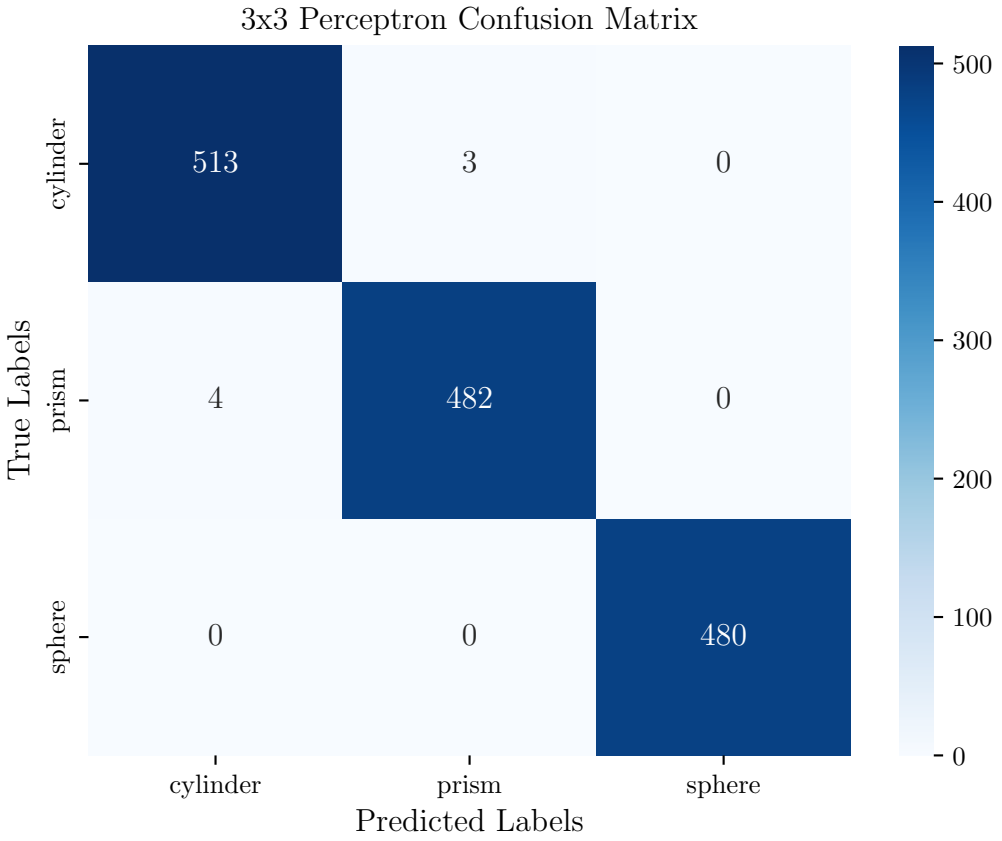


Figure 4.5: Perceptron Confusion Matrix.

Finally, a 5-fold cross-validation was performed. The model was trained five times, but with different parts of the training set each time, which would make the model more robust especially when the dataset is quite small. Again, this method is also a way of validating the reliability of the single training case in Table 4.1. Both the linear and polynomial kernels appear as good candidates for classifying the data when five-fold cross-validation is applied. Perhaps the greatest observation of all for this model is that the introduction of windowing confused the model instead of making it more certain. What appeared to be a model of almost 100% certainty was actually on average between 70-80% and the high accuracy was more case specific than it first appeared.

In addition to the model results themselves, an evaluation of the data separability was conducted. The first step to doing this was taking the augmented input data, and reducing its dimension from a high number to two dimensions using different methods. The first method, shown in Figure 4.6, uses the principal component analysis (PCA), which aims to capture the directions with the most variance in the data. The values on the x-axis capture the largest possible variance, or spread, in the data. The values on the y-axis capture the next largest variance, orthogonal to the first. Furthermore, the PCA is an unsupervised method and no classes are pre-defined before the analysis is done. Since this method shows similar data as

Table 4.1: Model Accuracy Comparison.

		Linear	Polynomial	RBF
Test Accuracy	Windowing	0.97	0.97	1.00
	No Windowing	1.00	1.00	1.00
Final Train Accuracy	Windowing	1.00	1.00	0.33
	No Windowing	0.95	0.96	0.88
5-Fold Cross-validation	Windowing	0.79	0.71	0.37
	No Windowing	0.95	0.96	0.81

clusters the conclusion is that according to this analysis, there is some natural separation in the data. The second method, shown in Figure 4.6 to the right, uses the linear discriminant analysis (LDA). This analysis method aims to maximize the separation between different known classes while minimizing the variance within each class. The Figure clearly shows that the data can be separated, which should make it close to, or fully linearly separable. This confirms why the simple models, such as the perceptron, are doing well with the classification. If the data had not been linearly separable, the models would likely have been less successful.

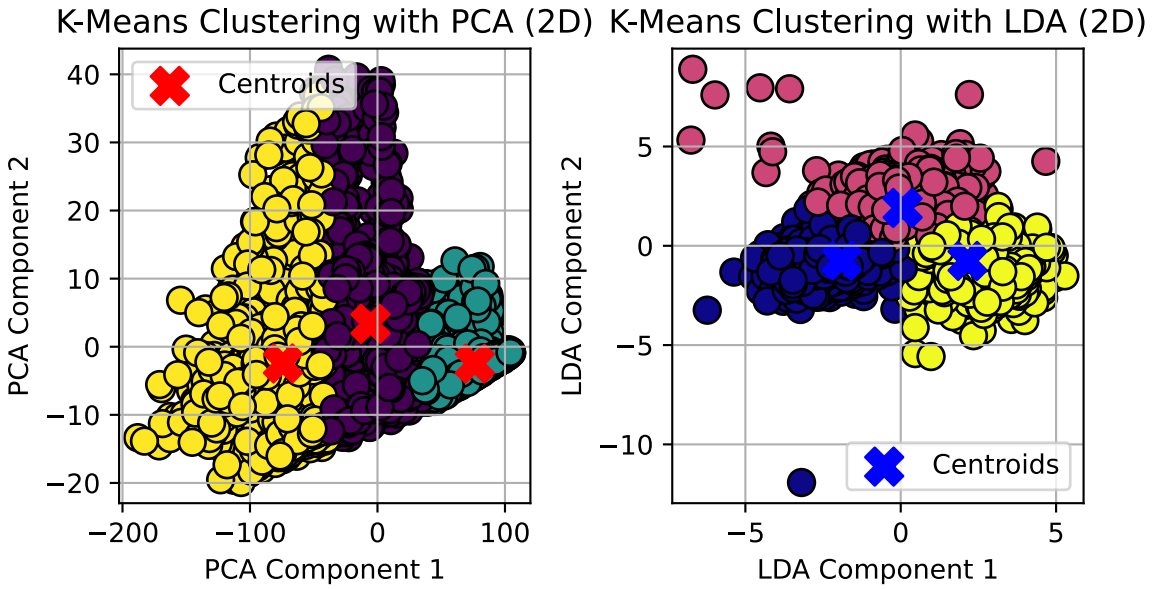


Figure 4.6: PCA and LDA plots with K-means clustering.

4.2 Force Prediction

The following section presents the results from the force prediction models, including four black box models and one glass box model for a final comparison.

4.2.1 Deep Learning Models

The force prediction model was, as mentioned, trained in four different ways. Again, they were trained with different input and output data types and shapes. These were the use of windows while predicting a single value at a time, the use of windows while predicting a series, the use of all input data up to a certain time step while predicting a single value, and a regression model predicting the full series at once. Out of the four, the training accuracy of the first method appears in Figure 4.7. The x-axis shows the training epochs, while the y-axis displays the accuracy in a percentage of all samples in the dataset. This accuracy refers to the predicted value or values being within a certain margin of error of the actual value. For most of the models, this allowed error is set to 5%. Both the training and validation accuracies converge after around 12 epochs and does so slightly under 90%. There is no direct indication of over-fitting present as none of the curves diverge from the other or show any unexpected behaviors.

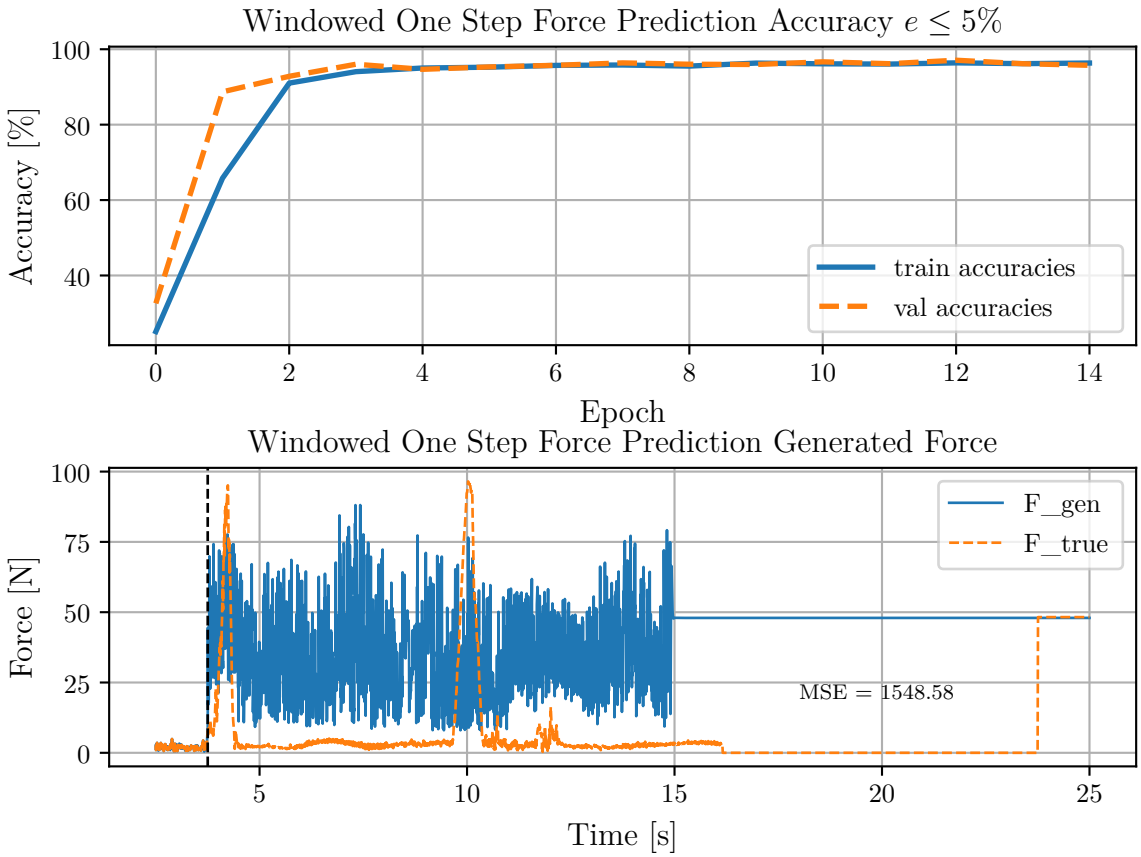


Figure 4.7: GRU model training one step forward prediction with windows 5% allowed error.

Even though the first model is able to output predicted values at a reasonable accuracy, the model was not able to handle the data when it was fed back in a closed loop. The full force was to be generated using an iterative method where the newly predicted force value should be included in the model input to predict the force in the next time step. Thus, the predicted forces became extremely volatile as shown in Figure 4.7. The cause of this behavior can be debated, but in theory, if the previously predicted values are good enough to be recognized by the model then the predicted forces could be more accurate. This could simply indicate that the model itself must achieve a better one step accuracy before it can be used in a feedback loop. The Mean-Square Error (MSE) of the prediction was 1549 for the first model.

Moving on to the second model, both its training accuracy and predicted forces appear in Figure 4.8. The model itself was trained to predict more than one value at a time, but only the first predicted value was fed back into the model. Supposedly, this was meant to give better results as the model has more knowledge of the future forces. In the upper figure, the accuracy during training can be observed and the model could only converge to an accuracy of just under 60% with an allowed error of 5%. This accuracy is low compared to the previous model but still produces better results than what was achieved by the one-step prediction model in Figure 4.7.

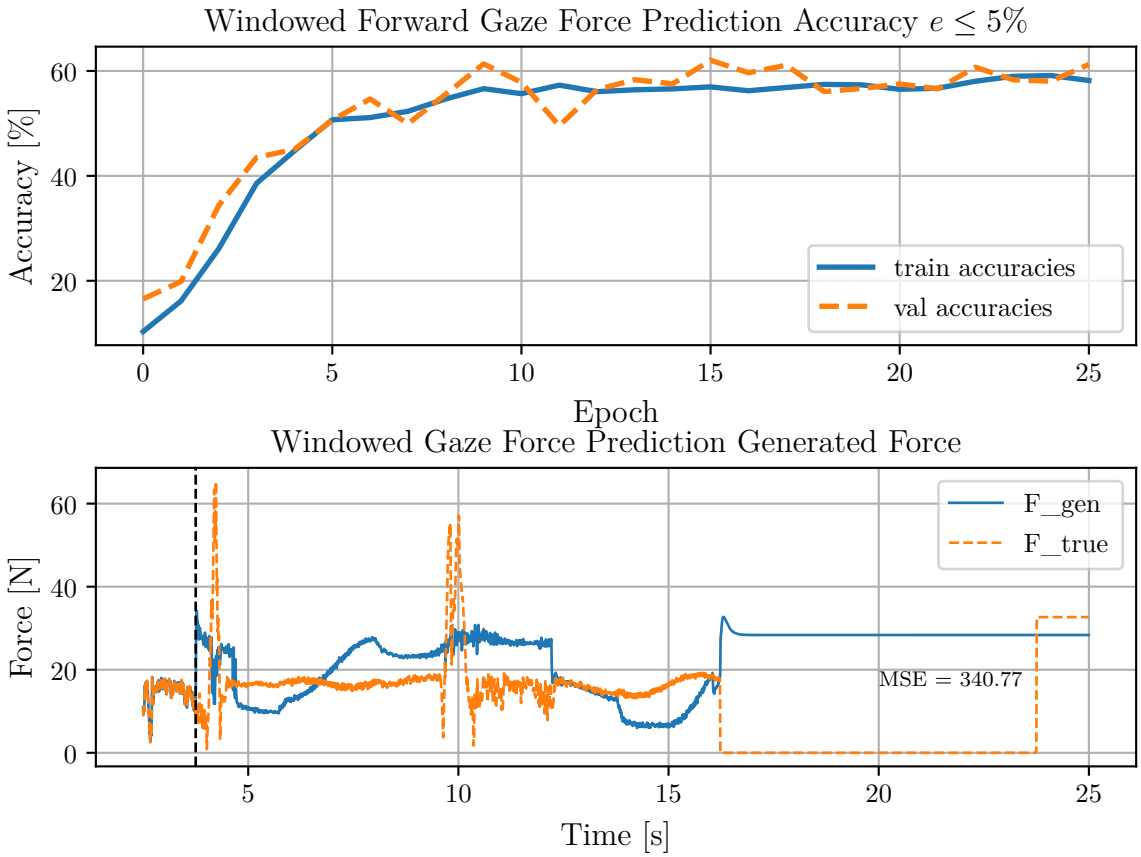


Figure 4.8: GRU model training gaze forward prediction with windows 5% allowed error.

The lower plot in Figure 4.8 shows the generated force from the prediction start time, which was at around three seconds. The predicted force does not diverge from the true force, however it could not predict the data peaks. Additionally, the MSE lies at around 340, which is still too high for the model to be considered successful since the average error is around 20N. Lastly, the gaze model managed to avoid high-frequency data fluctuations which were present in the one-step model.

The third model results are presented in Figure 4.9. The upper plot shows that the model converged to an accuracy of about 3 %, which indicates that the model failed to learn the patterns of the training data. Thus, the lower plot becomes redundant in this case, but is still shown for completeness.

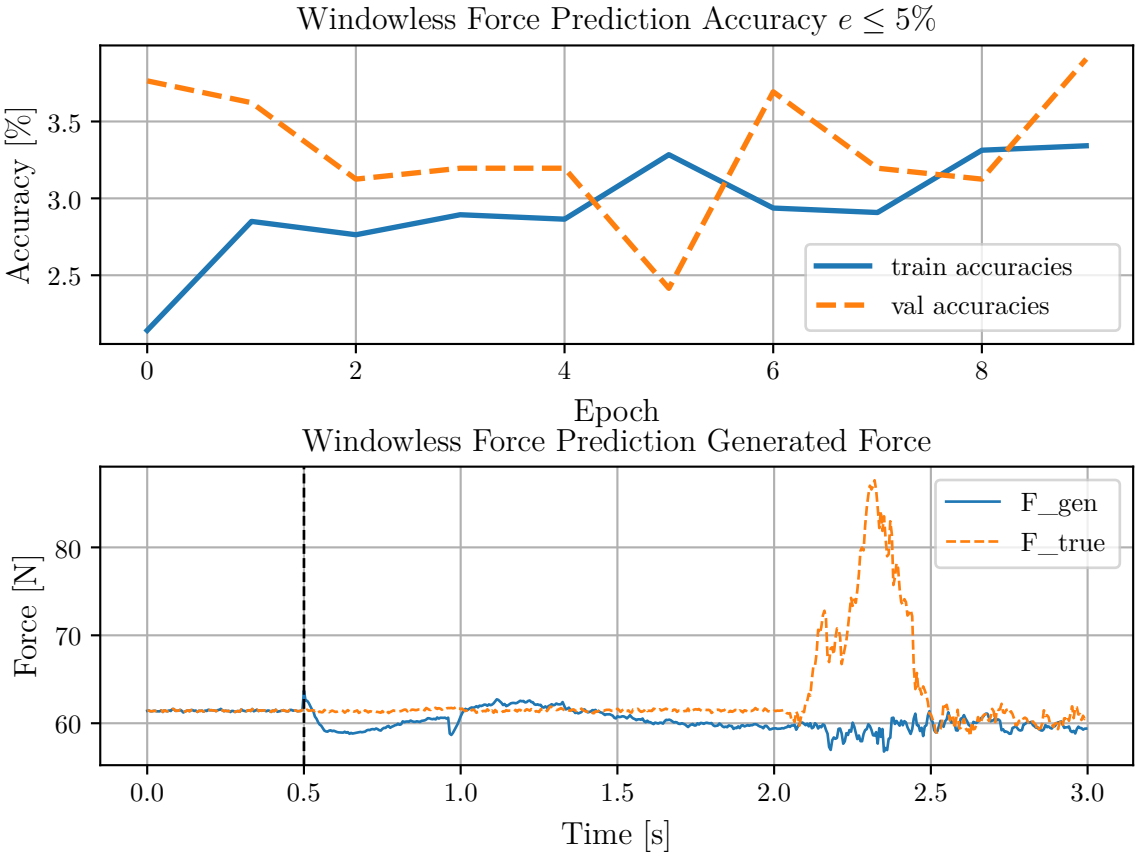


Figure 4.9: GRU model without windowing with one-step prediction.

The results from the fourth deep learning model, which is a simplified method compared to the three previous methods, appear in Figure 4.10. The top left plot shows the mean square error, which decreases with each epoch. This decrease is desired as it means that the model manages to minimize the model error. Additionally, the other plots show the model's actual performance when generating forces. The difference between the three force plots is that they each correspond to a separate particle type. As an example the top right plot shows the generated forces when handling cylinders. From all these plots it appears that the model manages to vaguely follow the trend of the experimental data. However, the model acts conservatively and smooths out the peaks, likely because staying close to the force mean

value minimizes the error more than making a large jump would do. Simply put, the model avoids large value jumps as it risks increasing the error significantly. Lastly, it is observed that the prism case seems to generate the most accurate result in terms of peak matching. The sphere case is shown to have the lowest error of 29. Thus, the question of which of these two results is better arises.

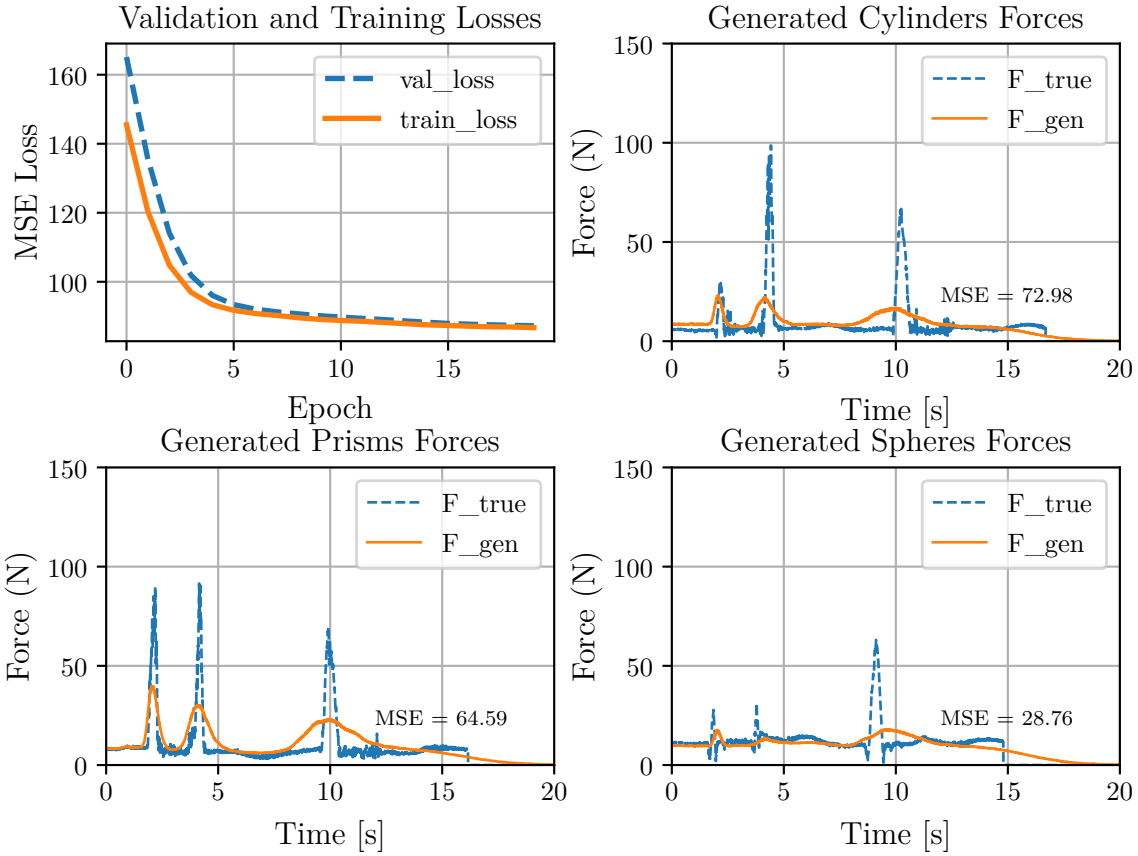


Figure 4.10: GRU model training and generated forces.

4.2.2 Interpretable Model

The interpretable models used for force prediction ended up applying the same input data as the deep learning model in Figure 4.10. The reason behind this is that the other models could not produce output forces that were able to mimic the true forces. Essentially, they were deemed inadequate and were not evaluated further in this project, also because of the time constraint of the project.

The perceptron model appears in Figure 4.11. The figure displays the model's training performance on the top left. In this case, the model tries to minimize the mean square error between the generated graph and the experimental data. The other three plots show the predicted forces plotted alongside the experimental forces. Each plot also corresponds to how the model tries to predict the resulting forces based on the particle type. As stated in the methods chapter, the inputs to this model were the particle types as well as the positions, velocities, and accelerations for the full time-series.

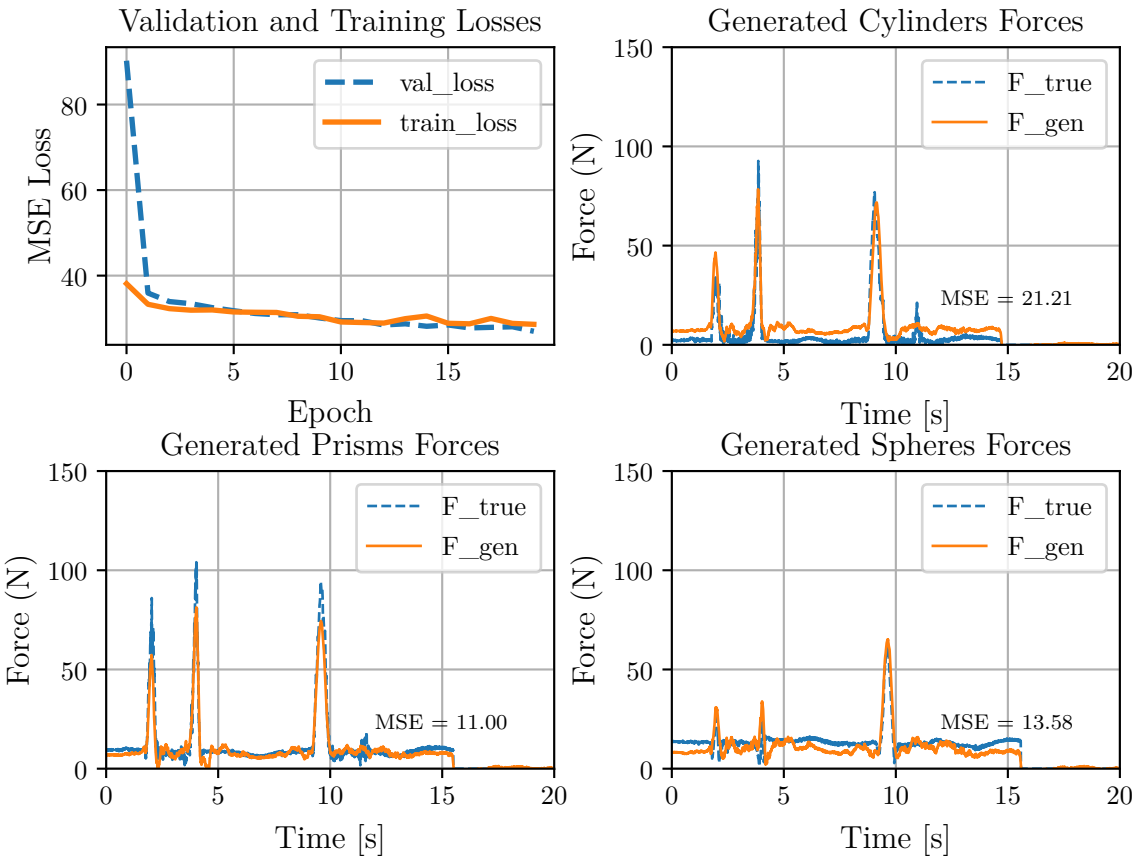


Figure 4.11: Regression model training and generated forces.

The best-performing models for particle classification and force prediction were the following. For particle classification, the GRU model achieved a test accuracy of 91%, with minimal confusion between particle types. The perceptron model predicted the particle shapes almost perfectly, reaching an accuracy of 99.53%, while providing clear visualizations of feature importance. The SVM models also showed strong results, particularly with linear and polynomial kernels, achieving near-perfect accuracy when windowing was not applied. For force prediction, the GRU models again stood out. The single-step windowed GRU model reached 90% accuracy but struggled when generating forces through the feedback method. The multi-step windowed GRU model had lower accuracy at 60%, though it generated smoother predictions with poor peak matching. The windowless GRU model demonstrated the best performance in capturing force trends, despite its conservative peak estimates. The perceptron model also showed promise in force prediction, delivering performance comparable to the GRU models while maintaining interpretability. However, the perceptron model lost the likeness to how forces are simulated in Demify® due to its lack of a feedback loop. In summary, the GRU models excelled in both classification and force prediction, with the perceptron model providing valuable insights into feature contributions.

4.3 Simulation Results

This section presents data that were generated using DEM simulations. The main purpose of this data is to enable deeper and more reliable evaluations of the force models' performances. In Figure 4.12 all of the simulated forces are plotted in the upper figure. The dashed line represents the force mean at each time step while the red area represents the standard deviation of the data. Finally, the purple area simply represents the largest sample force value found at each time step. Furthermore, both plots include all data points available. However, the simulation plot consists of much fewer samples compared to the experimental plot which consists of approximately 10 000 samples. The reason for this is that each simulated force was generated with different friction parameters. This was done simply because the real-world friction between particles as well as the friction between particles and the tool are unknown. Consequently, two variables were considered while generating the simulated forces. These variables are pp or particle-particle friction and pt, particle-tool, and friction.

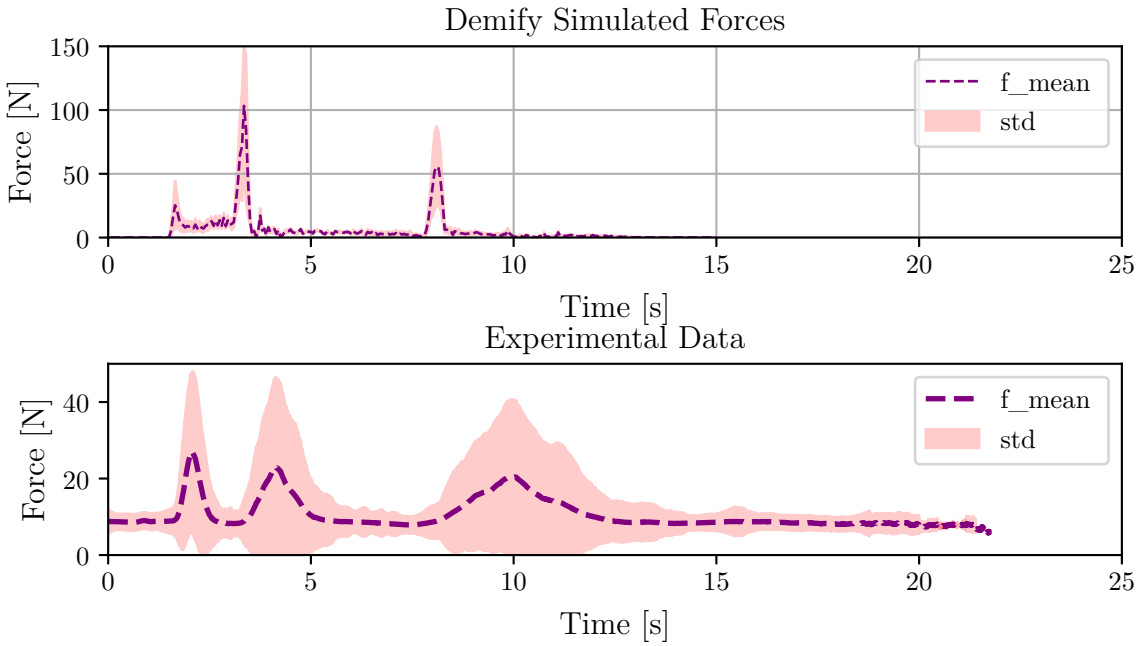


Figure 4.12: Distribution of Simulated and Experimental Data Over All Particle Shapes.

Figure 4.13 only includes data belonging to a certain particle shape. In this case, only prism samples are considered for both simulation and experimental data. The three marked points are the mean values of the maximum peak forces calculated from the experimental data. The blue graphs are the simulated forces for the prism shaped particles. Considering the friction values, an observation is that the pp friction seems to be the dominating variable.

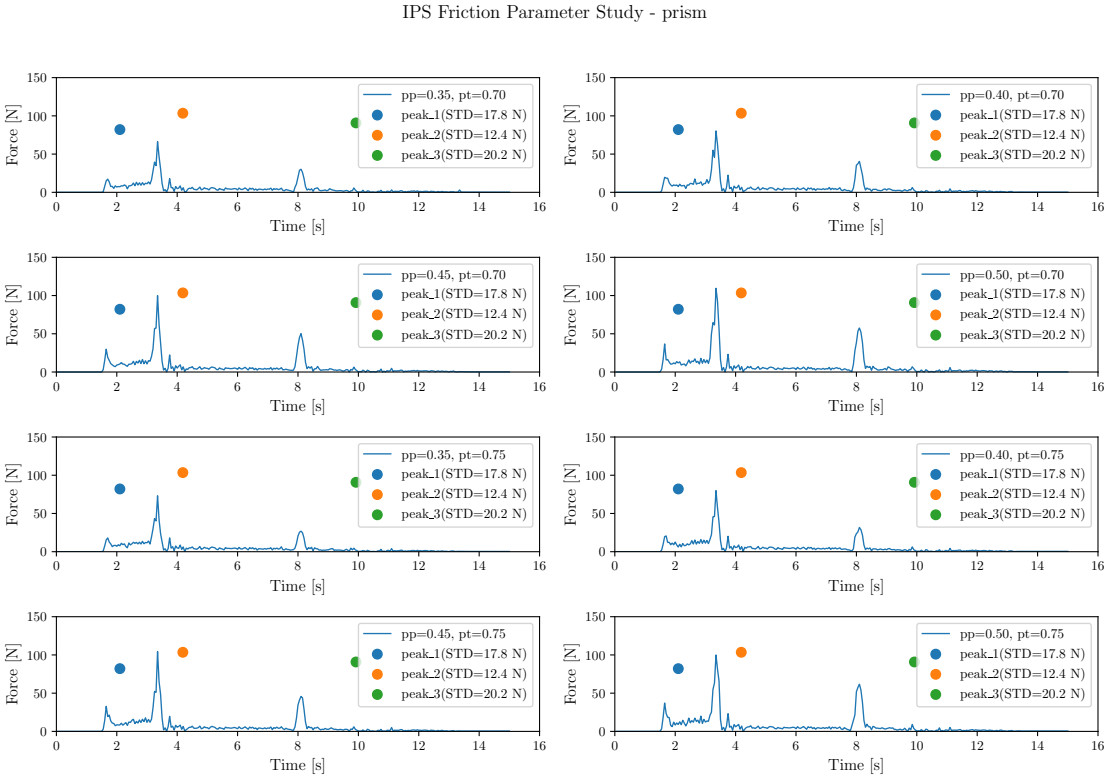


Figure 4.13: Simulated Data with Varying Friction and Mean Peak Values.

All of the simulated data was generated using frictions ranging from 0.3 to 0.8 with intermediate steps of 0.05. Another observation is that the simulated forces appear to spike earlier than the experimental forces. This might seem strange considering that the robot path in question is directly imported to Demify® from the robot itself. However, this is likely because the robot has to pause in between runs so that the audio is saved for each sample before starting the next run. This could also be caused by the varying speed of the experiments since the simulations are based on a single selected speed. Only having the robot path does not take this delay time into consideration.

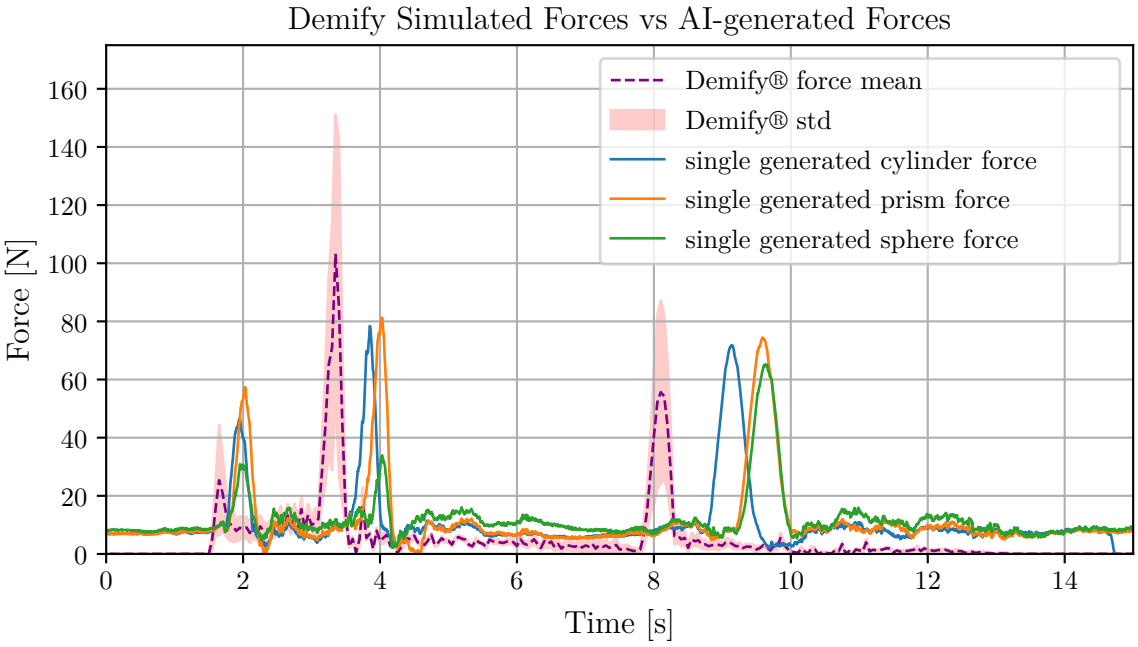


Figure 4.14: Simulated and AI-Generated Data Comparison.

In Figure 4.14 the Demify simulated forces are plotted along with three AI-generated force predictions. These force predictions are the same three found in 4.11 with each one of them corresponding to a single particle sample. The AI-generated forces are slightly delayed compared to the simulated forces but follow the same trend in spike magnitudes. Another observation is that the generated forces seem to match the mean values found in the parameter study for the simulated data well. Although the first spike seems to produce a larger max value in the AI-generated data compared to the simulated data. Overall, the generated data resembles the experimental data, which is expected since they were trained on that data. Furthermore, this also means that the generated data displays the same flaws as the experimental data e.g., not starting at zero force and time delays.

5
Discussion

5.1 Interpretation

5.1.1 Particle Classification

In the case of particle classification, the interpretable models outperformed the more complex deep-learning models. This might be because the data is close to or is linearly separable as observed in Figure 4.6. Therefore, it should be easily classified by data division done by hyperplanes, like the SVM, or by linear models like the perceptron. The high number of weights in the GRU-network could instead introduce over-fitting or a too high complexity to converge smoothly.

The SVM produced mostly good results regardless of which kernel combination was utilized, with the exception of the RBF kernel. When it comes to the degree of model interpretability, the kernel largely determined how easily the model parameters can be understood. The linear kernel was the most interpretable and comparable to the perceptron, which had the same number of parameters. Next, the polynomial kernel introduces some non-linearity which is harder to interpret. However, the polynomial boundary can still be plotted in a reduced 2D space, which makes it more interpretable than a typical black box model. Lastly, the RBF kernel could not be considered a truly interpretable model because of its high dimensionality.

The GRU model showed promising results and the accuracy convergence during training was relatively stable. Furthermore, the model did not seem to become overfitted to the data since both the training and validation accuracy increased at the same rate. As mentioned, the training time was a limiting factor, but the model still converged. Thus, more training time might not necessarily improve the model.

In order of most to least successful classification model comes the perceptron (99-100%), the polynomial SVM (96%), the linear SVM (95%), and the GRU (91%). Both the perceptron and linear SVM are linear classifiers that are preferably used with linearly separable data. Since both these models achieved good results, it suggests that the data is linearly separable. This makes more complex models unnecessary as they provide little to no improvement in exchange for longer training times. Although the perceptron showed a high accuracy, it was not higher than the initial test accuracies achieved by the SVMs before the five-fold cross validation was performed. This suggests that the perceptron too should be trained using folds before deciding which model performs better. Among the SVM models, the linear

kernel is likely a better choice than the polynomial, simply because of the added interpretability compared to the slight accuracy decrease. This is because, model interpretability provides valuable insights that enable a more confident and reliable evaluation of the model's performance.

Based on the previous reasoning, the best performing model would either be the linear SVM or perceptron, but more research is required to determine which model truly performs best for the task. The two models are similar and share the same number of features. However, they differ significantly in their approaches. The perceptron aims to minimize a classification error by adjusting weights based on misclassified examples, whereas the linear SVM tries to maximize the margin between classes using a predefined boundary shape. Additionally, the perceptron is more sensitive to outliers and may fail to converge if the data is not perfectly separable, while the linear SVM is more robust due to its soft margin approach. However, the SVM required a longer training time compared to that of the perceptron as the data could not be loaded dynamically. Instead, the whole dataset had to be loaded into memory before training, which added additional time. Furthermore, it is worth noting that the overall performance during the training process of the SVM was much more difficult to track due to the lack of epochs-based training. Some strengths of the SVM results were the following.

- The linear and polynomial SVMs have high training accuracies even when trained with five folds.
- The five-folds reduce bias towards a specific subset of the dataset, making the results more reliable.
- The linear kernel SVM has the same number of extractible weights as the perceptron.

As mentioned, the interpretable models were the best performing, which is quite surprising considering that deep networks are often the first attempt at a solution to be applied to problems. This proves that there are cases where simpler models can outperform more complex models. Therefore, it would be good practice to consider testing simpler methods before turning to deep networks in cases where problem complexity is uncertain or known to be low. The training time and interpretability of a simple network are valuable and it is a waste to discard such helpful features lightly. As an example of training time differences, the perceptron was able to train for 500 epochs in 30 minutes while the GRU network could take anywhere from 1-3 hours to run ten epochs. The SVM would take less time than the GRU network, but more time than the perceptron depending on the selected kernel.

5.1.2 Force Prediction

In the force prediction case, one of the main tasks was to compare AI-generated forces to simulated forces as a form of model validation. Both methods also had to cover the same loading scenario. Furthermore, the AI method was meant to mimic the DEM-based way of simulating forces, i.e. predicting the forces dynamically for each time step. This, however, was more difficult than expected because the feedback-based models did poorly even though they produced high accuracy

results in one time step forward predictions. Although this was the case, the models did not seem to diverge, although they still showed little to no pattern recognition. This behavior suggests that the forces, which were fed back into the model, were too different compared to the true force, thus the model was unable to predict the forces properly. In contrast to the expectations, the window-based models increased the uncertainty in the predictions and did not increase the accuracy at all. This led to a decision to exclude windows in the models that had not yet been trained, i.e. models three and four. The windows were replaced with the whole data series up to a chosen time step. This was the case for the third GRU model.

Another concern arose regarding the selected features that were used as input data. The problem with these features was that they also had to be updated in each loop. The input data fed back into the network consisted of features such as the mean force, slope of the window and frequency related features. This update likely only confused the model, which led to the decision to exclude these features. The only utilized data thereafter were forces and position-related data, and this was an issue only in Figure 4.7 and Figure 4.8.

The third model, seen in Figure 4.9, proved to be more stable than the two window-based models, however, with the drawback that the force peaks were not properly predicted. The derivative of the force data was added as an input feature to this model, which aided stabilization in the prediction results. However, the model still used feedback as a way to generate a full-force series. Now, most of the previously mentioned model configurations could not be considered greatly successful. It is possible that this method of feedback-based data generation could work either with a different set of input features or with a different network altogether. As it stands, the results suggest that the feedback-based data generation was no suitable method for this specific application.

The fourth model was the most successful out of all GRU models for force prediction, but with a trade-off. The model can be considered simpler compared to the other models, and loses the dynamic feature that the other GRU models had. The model essentially works as a more complex curve fit which can adapt based on position and particle type data. However, one of the larger drawbacks with this model is that it must be given positions, velocities, and accelerations for all time steps. A concern is of course that both the feedback and the input and output format was changed at the same time, making it hard to know what exactly is making the model produce better results. This should probably be investigated by testing a GRU model with the model two inputs while doing full predictions. Another model should also be evaluated, which uses the fourth model's inputs while also generating forces using a loop. Because the fourth model was the best performing out of the four GRU models, a glass box model was made using the same input data format.

The glass box model, which was perceptron-based just like in the classification case, showed great results. The mean square error was, overall very low in the perceptron case compared to many of the GRU cases. The outcome is in line with the

expectations since regression models are commonly applied to curve fitting tasks. The perceptron has been transformed into a regression model by making it predict a time series instead of classes and by continuing to exclude the use of an activation function. Regression models are commonly used for curve-fitting tasks after all.

In conclusion, the best-performing model in this case was the perceptron. There is still much potential for improvement, and perhaps a fully data-driven approach is not the most suitable for the task considered in this project. Instead, it might be better to focus on a combination of data and equations in the models, for example through a physics-informed neural network, or even introduce some other aspects of interpretability. Although the more complex models showed relatively low accuracies, they performed well for one-step predictions, however, they failed to make accurate predictions once in the feedback loop. The particle type could possibly have been included in the earlier models, but was excluded since the possibility was realized too far into the project.

5.2 A Comparison between Simulations, Experiments, and Machine Learning models

The larger aim of this section is to compare the final performance of both simulations, experiments, and the best-performing AI model. Mainly, the goal is to answer how well the AI-generated force predictions compare to the Demify® simulated forces. The answer is that they are largely similar, both appearing to have force peaks of around 100 N and mostly at the same time. The large max peaks in the simulated data observed in Figure 4.12 are largely due to the varying friction parameters. In reality, and as in the case of the experimental and AI-generated forces, the peaks should form consistently at the very least under 150 N in real-world scenarios. This means that some of the frictions used in the simulations were too high to accurately describe the experimental data. Another difference between the AI-generated and simulated data is that the generated data does not start at zero. This is likely because the experimental data includes the gravitational force from the bucket attachment at the end of the robot arm.

Next, some aspects of the experimental data should be discussed. First, the data shows some variations in force peak placement, which is largely due to there being three particle types and that the robot execution speed was adjusted for each run. However, one might argue that the data is still very case specific and bound to a specific robot path. This is something that needs to be considered as a future improvement where the addition of more diverse cases would benefit the model's adaptability.

Another comparison to be made is between the simulated and AI-generated data. Both of these were plotted in Figure 4.14. From the plot, it is clear that the regression based force prediction closely matches the simulated force prediction. The peaks of the AI-generated data are mostly inside of the simulation standard devia-

tion. However, it is unlikely that the regression model would generate values close to the max values of the simulated data. The conclusion, which also answers one of the main questions of the project, is that both force predictions are very similar. However, the choice of friction in Demify® does affect the final results significantly. In conclusion, the regression AI-model predictions show very similar results to that of the DEM-based simulation software.

5.3 Further Improvement

Although many topics were covered in this project, there is always room for improvement. General recommendations for such improvements are presented in this section. First, the data collection itself could be made more efficient, and it would be beneficial to see if ROS can be used for this application instead of the universal robots real-time data exchange (RTDE) extension. Using RTDE is still sufficient, but ROS could be beneficial due to ist many integrated packages and extensive knowledge base. This could be worth exploring. The reason for ROS being excluded was that RTDE was simple to set up and required little additional knowledge like ROS does.

Next, the particle classification case could be evaluated further using the five-fold method to train the perceptron as well as neither the SVM nor the perceptron could be considered better for the task.

Secondly, the dataset could benefit from more variation as it lacks both the use of different paths and scenarios. Paths refer to the use of the same rig but with varying robot movement and goal positions, while different scenarios refer to the use of different rigs. This is mainly to increase the robustness of the AI-models and make them more adaptable to different real-world scenarios.

Next, a different force prediction model could be used. The initial models were not very successful, while the chosen model lacked complexity and the desired way of handling the data. Perhaps a transformer or simply a change in the input data format may be helpful. The fourth model should be evaluated more by also testing it with a feedback loop. Other than that, interpretable models were helpful in this project. Perhaps a combination of AI and an equations-based solution could help the model learn and generate real-time force predictions better.

Lastly, since the simulation results have been shown to be able to represent the experimental data quite well, this data could possibly be used to extend the dataset.

6
Conclusion and Future Work

The machine learning model, specifically, the perceptron-based regression model, produced force predictions that closely mirrored those simulated by DEM. The AI-generated forces and the DEM forces displayed similar force peaks and trends, validating the model's accuracy. However, the choice of friction parameters in the DEM simulations significantly influenced the results. Additionally, the AI-generated data showed consistent force trends, but did not start from an initial force value of zero, likely due to how the data was collected from the robot.

In the classification task, simpler models, namely the linear SVM and perceptron, outperformed more complex deep learning models. The perceptron in particular achieved the highest accuracy, which was close to 100%. Both the perceptron and linear SVM worked well with the linearly separable data, suggesting that complex models are not necessary for this task. This, however, should not take away from the fact that the deep model still managed to achieve a high accuracy, but at the cost of training time. The choice of model ultimately depended on the balance between performance and interpretability.

In both classification and force prediction, interpretable models like the linear SVM and perceptron outperformed deep neural networks to a certain degree. While the force prediction deep learning models showed potential during training, they struggled when trying to generate accurate dynamic predictions in force prediction tasks. In contrast, the perceptron-based regression model was the most successful for force prediction, capturing force trends effectively and demonstrating the advantage of simpler, interpretable models. These findings emphasize that simpler models can outperform more complex ones when the problem's complexity is low, highlighting the value of interpretability in model evaluation and real-time applications.

Although the project achieved promising results, there are areas for improvement. Enhancing the data collection process, incorporating more diverse scenarios, and experimenting with alternative force prediction models, such as transformers, could optimize the models' performance. A combination of data-driven methods and equation-based solutions might offer better results in real-time force prediction. Additionally, making use of the simulated data to extend the dataset could further enhance the models' adaptability and prevent over-fitting.

Bibliography

[1] M. Bilal et al. "Speech Emotion Recognition in Pakistani-Accented English Language". In: *2023 17th International Conference on Open Source Systems and Technologies (ICOSST)*. IEEE, Dec. 2023, pp. 1–4. DOI: 10.1109/icosst60641.2023.10414231.

[2] H. D. Block. "The Perceptron: A Model for Brain Functioning. I". In: *Rev. Mod. Phys.* 34 (1 Jan. 1962), pp. 123–135. DOI: 10.1103/RevModPhys.34.123. URL: https://link.aps.org/doi/10.1103/RevModPhys.34.123.

[3] J. Cao et al. "Excavation equipment classification based on improved MFCC features and ELM". In: *Neurocomputing* 261 (2017). Advances in Extreme Learning Machines (ELM 2015), pp. 231–241. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2016.03.113. URL: https://www.sciencedirect.com/science/article/pii/S0925231217302102.

[4] K. Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*. 2014.

[5] J. Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE]. URL: https://arxiv.org/abs/1412.3555.

[6] S. Clarke. "Robot learning for manipulation of granular materials using vision and sound". Master's thesis. Pittsburgh, USA: Carnegie Mellon University, 2019.

[7] P. A. Cundall and O. D. L. Strack. "A discrete numerical model for granular assemblies". In: *Géotechnique* 29.1 (1979), pp. 47–65. DOI: 10.1680/geot.1979.29.1.47.

[8] R. Dey and F. M. Salem. "Gate-variants of Gated Recurrent Unit (GRU) neural networks". In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2017, pp. 1597–1600. DOI: 10.1109/MWSCAS.2017.8053243.

[9] I. El Naqa and M. J. Murphy. "What Is Machine Learning?" In: *Machine Learning in Radiation Oncology: Theory and Applications*. Ed. by I. El Naqa, R. Li, and M. J. Murphy. Cham: Springer International Publishing, 2015, pp. 30–31. ISBN: 978-3-319-18305-3. DOI: 10.1007/978-3-319-18305-3_1. URL: https://doi.org/10.1007/978-3-319-18305-3_1.

[10] W. J. von Eschenbach. “Transparency and the Black Box Problem: Why We Do Not Trust AI”. In: *Philosophy & Technology* 34.4 (Sept. 2021), pp. 1607–1622. ISSN: 2210-5441. DOI: 10.1007/s13347-021-00477-0.

[11] A. Graves, A.-r. Mohamed, and G. Hinton. “Speech recognition with deep recurrent neural networks”. In: *arXiv e-prints* (Mar. 2013).

[12] S. Han et al. “Classification and regression models of audio and vibration signals for machine state monitoring in precision machining systems”. In: *Journal of Manufacturing Systems* 61 (2021), pp. 45–53. DOI: 10.1016/j.jmsy.2021.08.004. URL: https://www.sciencedirect.com/science/article/pii/S0278612521001667.

[13] Y. Huang and Y. Xiong. “A Study on the Prediction of Excavation Forces Based on Artificial Neural Networks”. In: *Journal of Engineering Science and Technology* 12.7 (2017), pp. 1895–1908.

[14] I. Kim, Y. Kim, and S. Chin. “Deep-Learning-Based Sound Classification Model for Concrete Pouring Work Monitoring at a Construction Site”. In: *Applied Sciences* 13.8 (2023). ISSN: 2076-3417. DOI: 10.3390/app13084789. URL: https://www.mdpi.com/2076-3417/13/8/4789.

[15] M. Kumar, T. Ekevid, and W. Löwe. “Operator model for wheel loader short-cycle loading handling”. In: *Automation in Construction* 167 (2024), p. 105691. DOI: https://doi.org/10.1016/j.autcon.2024.105691. URL: https://www.sciencedirect.com/science/article/pii/S0926580524004278.

[16] M. Kumar et al. “Driving Pattern Classification for Wheel Loaders in Different Material Handling Using Machine Learning”. In: *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. 2023, pp. 283–290. DOI: 10.1109/ITSC57777.2023.10422375.

[17] H. Lee et al. “Unsupervised feature learning for audio classification using convolutional deep belief networks”. In: *Advances in Neural Information Processing Systems*. 2009, pp. 1096–1104.

[18] S. Liu et al. “LSTM-based aerodynamic force modeling for unsteady flows around structures”. In: *Wind and Structures* 38.2 (Feb. 2024), pp. 147–160.

[19] A. McCallum. *MALLET: A Machine Learning for Language Toolkit*. Available at http://mallet.cs.umass.edu. 2002.

[20] M. Minsky and S. A. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

[21] B. Nye et al. “journal for numerical and analytical methods in geomechanics”. In: (2014), p. 38. URL: https://doi.org/10.1002/nag.2299.

[22] S. Paneru and I. Jeelani. “Computer vision applications in construction: Current state, opportunities & challenges”. In: *Automation in Construction* 132 (2021), p. 103940. ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2021.103940. URL: https://www.sciencedirect.com/science/article/pii/S0926580521003915.

[23] D. Peng et al. “Prediction of milling force based on spindle current signal by neural networks”. In: *Measurement* 205 (2022), p. 112153. DOI: https://doi.org/10.1016/j.measurement.2022.112153. URL: https://www.sciencedirect.com/science/article/pii/S0263224122013495.

[24] F. Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain [J]”. In: *Psychological review* 65.6 (1958), pp. 386–408.

[25] T. N. Sainath and C. Parada. “Convolutional Neural Networks for Small-Footprint Keyword Spotting”. In: *Sixteenth Annual Conference of the International Speech Communication Association*. Dresden, Germany, 2015. URL: http://www.isca-speech.org/archive/interspeech_2015/sainath15_interspeech.html.

[26] C. Schenck et al. “Learning robotic manipulation of granular media”. In: *CoRR* abs/1709.02833 (2017). URL: http://arxiv.org/abs/1709.02833.

[27] C. Schenck et al. “Which object fits best? Solving matrix completion tasks with a humanoid robot”. In: *IEEE Transactions on Autonomous Mental Development* 6.3 (2014), pp. 226–240.

[28] H. Soltau, H. Liao, and H. Sak. “Neural speech recognizer: Acoustic-to-word LSTM model for large vocabulary speech recognition”. In: *arXiv preprint arXiv:1610.09975*. 2016.

[29] N. Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[30] R. W. Sumner, J. F. O’Brien, and J. K. Hodgins. “Animating sand, mud, and snow”. In: *Computer Graphics Forum*. Vol. 18. 1999, pp. 17–26.

[31] Y. Tang et al. “Question detection from acoustic features using recurrent neural network with gated recurrent unit”. In: *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. 2016, pp. 6125–6129.

[32] C. Thon et al. “Multi-modal framework to model wet milling through numerical simulations and artificial intelligence (part 1)”. In: *Chemical Engineering Journal* 449 (2022), p. 137794. DOI: https://doi.org/10.1016/j.cej.2022.137794. URL: https://www.sciencedirect.com/science/article/pii/S1385894722030855.

[33] C. Thon et al. “Multi-modal framework to model wet milling through numerical simulations and artificial intelligence (part 2)”. In: *Chemical Engineering Journal* 450 (2022), p. 137947. DOI: https://doi.org/10.1016/j.cej.2022.137947. URL: https://www.sciencedirect.com/science/article/pii/S1385894722034337.

[34] O. K. Toffa and M. Mignotte. “Environmental Sound Classification Using Local Binary Pattern and Audio Features Collaboration”. In: *IEEE Transactions on Multimedia* 23 (2021), pp. 3978–3985. DOI: 10.1109/TMM.2020.3035275.

[35] S. L. Ullo et al. “Hybrid Computerized Method for Environmental Sound Classification”. In: *IEEE Access* 8 (2020), pp. 124055–124065. DOI: 10.1109/ACCESS.2020.3006082.

[36] T. H. Vu and J.-C. Wang. “Acoustic scene and event recognition using recurrent neural networks”. In: *Detection and Classification of Acoustic Scenes and Events*. 2016.

[37] W. Xie et al. “Prediction of construction cable forces of CFST arch bridge based on DNN”. In: *Structures* 61 (2024), p. 106012. DOI: https://doi.org/10.1016/j.istruc.2024.106012. URL: https://www.sciencedirect.com/science/article/pii/S2352012424001644.

[38] Y. Zeinali and S. T. Akhavan Niaki. “Heart sound classification using signal processing and machine learning algorithms”. In: *Machine Learning with Applications* 7 (2022), p. 100206. ISSN: 2666-8270. DOI: 10.1016/j.mlwa.2021.100206. URL: https://www.sciencedirect.com/science/article/pii/S2666827021001031.

[39] H.P. Zhu et al. “Discrete particle simulation of particulate systems: Theoretical developments”. In: *Chemical Engineering Science* 62.13 (July 2007), pp. 3378–3396. ISSN: 0009-2509. DOI: 10.1016/j.ces.2006.12.089.