Predicting the thermal profile of a system from the value of the lattice constant (rho) using Machine Learning techniques.

Methodology

Obtaining data

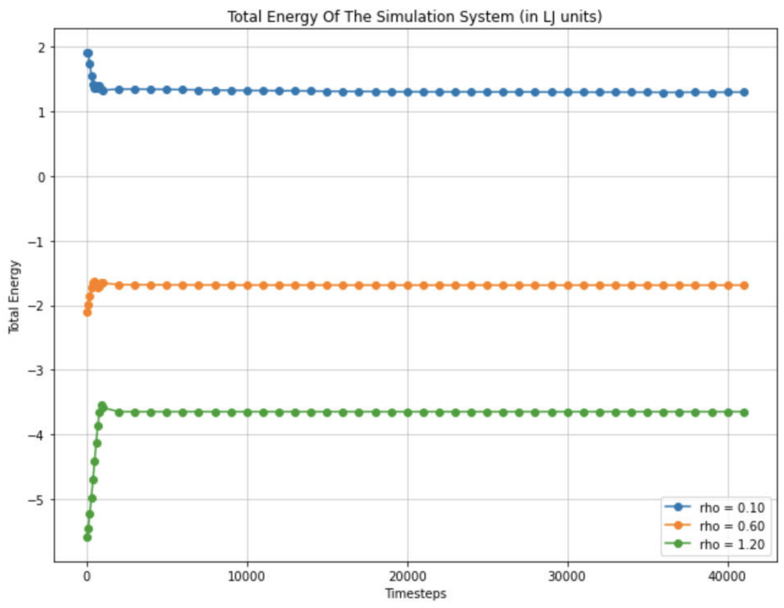
In this report, the data used to train the various Machine Learning (ML) models was generated through Molecular Dynamics (MD) simulations using the MD code; LAMMPS. The LAMMPS simulations were ran via a Secure Shell on the terminal, utilizing the capabilities of the 'Aristotle' supercomputer at UCL [1]. These simulations required an initial physical system to be set up, the details of this setup can be found in the appendices, where the variable being changed was the value of the lattice constant [2] (rho) which defined how closely atoms in the simulation box were packed together. The system had a face centred cubic lattice structure and the value of 'rho' ranged from 0.1 to 1.2 Nanometres. The aim of these simulations was to predict the thermal profile of the system (the temperature values on the z direction at each of the 20 chunks in the simulation box), this was achieved with the Muller-Plathe method which is a non equillibrium molecular dynamics method used in thermal conductivity prediction. The basis of this method is to split the simulation box into N layers (20 layers in this case) in the heat flux direction (z for these simulations) and to essentially swap the velocities of the hottest atoms in the outer (coldest) layer with the coldest atoms in the middle (hottest) layer therefore inducing a temperature gradient in the system [3] (conservation of energy and mass must be observed).

Checking for equillibrium

The simulations were run for 42,000 timesteps overall (each timestep being 1000 picosecond), with an initial 2000 timesteps to attempt to equilibrate the system and a subsequent set of two runs each 20,000 timesteps where the temperature of the system was observed. The obtained data files include `profile.mp`, which contains the chunk numbers and their respective temperatures at each timestep as well as `log.lammps` files that include how the other parameters of the system (total energy, E_mol, pressure, temperature, etc) varies over the simulation.

Then, equilibrium was checked to ensure the data files were appropriate for the later pass to the ML models for training. This equilibration check was achieved by creating plots of total energy (and temperature) of system against timesteps using `Python matli. The plots for the extreme values of rho (0.1, 0.6, 1.2 nm) are shown below. All rho values in the range 0.1-1.2 nm were checked for equilibration however it is unnecessary to include the complete set.

From the plots below we observe that the total energy stays unchanged as timesteps increase after a certain cut-off point (around 2000 timesteps). This provides evidence that the system has reached equilibrium and we are able to process it to be used in our ML models subsequently. The importance of the equilibrium check is to make sure the system is stable, and the corresponding temperature profiles are no longer fluctuating.



Extracting and processing data

After the equilibrium check, the data files need further manipulations. Particularly, the profile.mp files created from the simulations are of most interest as they contain the required input and output data for the ML models. The profile.mp files consist of the chunk number, coordinates, number of counts, and the temperature values for 42000 timesteps, among which we emphasize on the chunk number and the temperature values.

The python script overall_final.py contains the code to calculate the average of temperature values of each chunk in the last 10000 timesteps and then extract these average temperatures and their corresponding chunk numbers to a 'text' file using np.savetxt function. This ensures the training data we will use in the machine learning models is obtained from stable simulation systems so that we can accurately predict the temperature profile from an equilibrated system given a particular rho value.

Once extracted from the `profile_*.mp` files, the input ('rho' values) and output (temperature profiles) were loaded in from their respective text files using the `np.loadtxt` function to create an array containing each set. The shape of the input and output data is very important for ML models, especially neural networks due to the requirement for the dimensions of the input and output layers to match the respective data sets. As the input data contains only one feature ('rho') the data should have one column and then the number of rows will be the number of samples, in this case there are 56 total samples in the data set. The input data was processed using the `np.reshape` function to create a (56,1) dimensional array and similarly for the output as there are 20 different chunk temperatures the array was reshaped to be a (56,20) dimensional array.

To use the processed data to train and test various ML models the data must be split up into various categories, these include the training, cross-validation and testing data (varies depending on the types of model). For the XGBregressor model due to the incorporation of the `cross_val_score` function we simply divided the overall dataset into 80% training data and 20% testing data respectively. These are stored as `X_train`, `y_train`, `X_test` and `y_test` respectively. However, for NN, a further splitting was performed so that 80% of the data was still used to train the model, however now 10% was cross validation data and the remaining data was used to test the model (`X_val` and `y_val`).

Machine Learning models

The task of this project was to be able to predict the temperature values of chunks (positions) in a system from a single input parameter, 'rho', this is an example of a regression problem. This regression task has the unusual nature that there is only a single input parameter (defining the lattice structure) and the model must predict an entire profile conisting of temperature values for each of the 20 'z' chunks. This task, being a regression problem, has various ML models that may be suitable for implementation. It was concluded, after consideration of the points made in the thesis by Jana Bohm's:

> Application of Machine Learning Techniques to Material Science in order to analyze the Material Property-Structure-Process Relation.

it was concluded that XGBoost and Neural Network (NN) models are likely to give the best results. As the problem is a unique and complex regression model, the more complex NN approach may provide better results due to its structure. The models produced required various libraries imported into python, these included: Scikit-learn, Tensorflow, XGBoost, Pandas, Numpy and Matplotlib.

XGBRegressor models

Extreme gradient boosting (XGBoost) is an optimised decision tree based gradient boosting algorithm. It boosts the gradient descent process to find the global minimum by combining the predictions of lots of 'weak learners', and can be used in python for various ML tasks.

The specific type of model found to be suitable for the thermal profile prediction was the XBRegressor model, this model has various hyperparameters that must be tuned to produce the best model for the dataset. There are two main methods used for tuning of hyperparameters for these types of ML models, these are `RandomizedSearchCV` and `GridSearchCV`. These methods both take a list of hyperparameters as well as the range of arguments the parameters can take as an input and then produce a grid with possible combinations of the parameters, this can be done randomly or in a way defined by the user (Grid aproach). The method seen to have greater success for these regression type models is the `RandomizedSearchCV` which was found to give a higher score for hyperparameter tuning [4]. This method was therefore chosen to tune the hyperparameters for this model, the hyperparameters that were focused on included: `n_estimators` (the number of trees in the random forest), `max_depth` (the longest path between the root node and the leaf node, how deep each tree in the network can be), `eta` (the learning rate for the model), `subsample` (the percentage of training data used before growing trees, this prevents overfitting of the model) and `colsample_bytree` (the percentage of features used for building each tree). For each of the parameters, a wide range of values was initially used to identify the where the optimal values would be found and then this was then targeted more closely.

The accuracy of the model was tested as the mean squared error, calculated using the `cross_val_score` function to perform cross validation across the whole dataset with 10 different stratified samples, defined by `n_splits`, and repeated three times , defined by `n_repeats`, to ensure sufficient cross validation accuracy. The highest accuracy model found from the `RandomizedSearchCV` was then used to predict the temperature profile for a percentage of the overall dataset (test data) from their input values of 'rho'. This was then used to calculate the Mean Squared Error (MSE) of the model which also ensured the models accuracy. The predicted temperature profiles were plotted alongside the actual temperature profiles from the orignal data, this would further asses the accuracy of the model produced.

Neural network models

The other ML model used to tackle the temperature profile prediction was Artificial Neural Networks (ANN's), these are networks of nodes connected by neurones that is meant to immitate the structure of the brain. The software package used to implement these ANN's was Tensorflow and Scikit-learn, the initial task to create this model is to define the architecture of the neural network that will be most optimal for task.

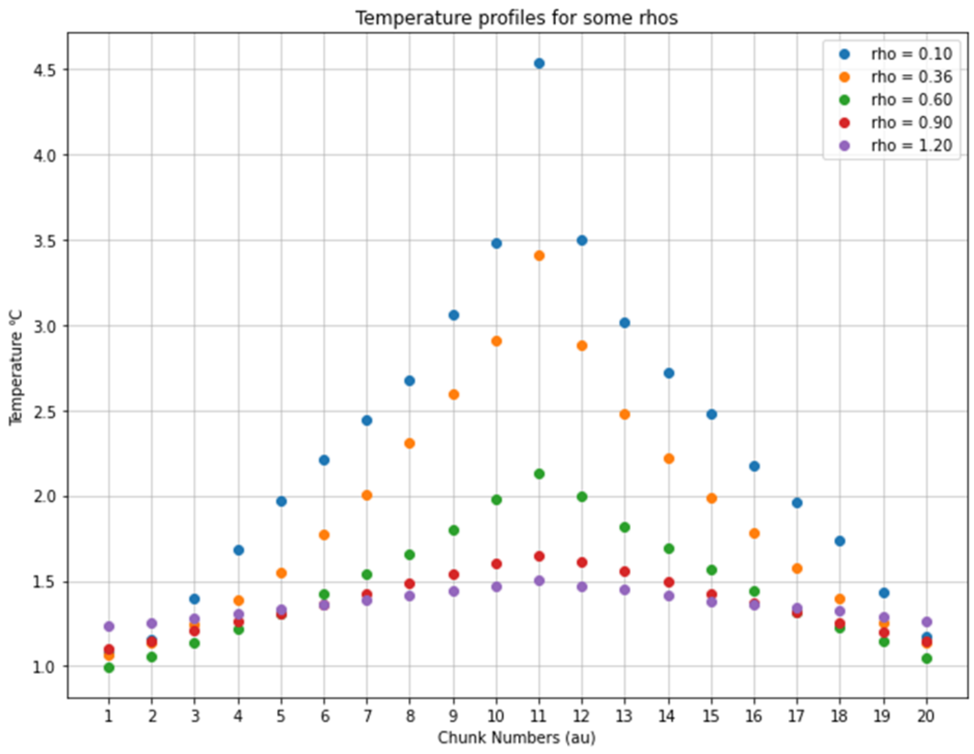
Every NN has at least 2 layers, the input and output layers which must match the structure of the input/output data used in the training. As described in the 'Extracting and Processing' section the input is a (56,1) array and therefore the input shape for the first layer is (1,) where '1' represents the number of features in the input (the only feature is 'rho'). Moreover, the output layer must match the shape of the output which is a (56,20) array and the output layer must contain 20 nodes (defined as `units` in the script) to produce the required output. Building upon the simplest possible NN structure, a NN with greater than two layers is known as a Deep Neural Network which has a number of 'hidden layers' that aren't involved directly in the input or output. The number and size (number of nodes) of hidden layers was altered and the resulting models' accuracy was tested to find the most optimal NN architecture. This was firstly done manually to get an idea of the types of structures that would provide the best results, from here the process could be automated using the `RandomizedSearchCV` method discussed in the previous section. It is concluded from the literature [6] that the number of neurons in hidden layers should decrease from the number of neurons in the input layer towards the number of neurons present in the output layer. Using this knowledge various possible network structures were tested to see which one produced the greatest accuracy.

Similarly to the hyperparameter optimization carried out for the XGBRegressor model described above there are various other parameters that can be provided for the NN which will effect the ability of the model to predict the thermal profile required by this task. The hyperparameters that were focused on in this study were: `learning_rate` (the rate at which gradient descent occurs, the size of each step), `epochs` (the number of times each sample is fed into the NN, one foward and back pass), `batch_size` (the number of samples given to the NN each `epoch`). Finally the activation function is an important parameter for ANN's, due to the nature of the task being a regression problem where the output can only have positive values (temperature can't be negative) the activation function used on each of the hidden layers was the `relu` (Rectified Linear Unit activation function, where the function is equal to zero if the input value is smaller than zero-denoted `max(0,x)`).

A similar process as described before with initially wide value ranges for the parameters was initiated which then allowed narrowing to find the best parameters for the most optimal model, this was again tested against the MSE metric to allow accurate comparison with the XGB model produced. Due to the separate cross validation data set produced in the data processing section, when the `model.fit` function is called to fit the model with the training data provided a cross validation loss, `val_loss` can be calculated along with the training loss, `loss`, from the model. These two metrics allow the model to be evaluated for under/overfitting. If the model is overfitted the trends would show that the `val_loss` is much greater than the `loss` value at the end of training however for the scenario where `val_loss` is lower than `loss` we may have underfitting. In the case where underfitting occurs we must consider whether the model is actually underfitting the data or if the reason for discrepancy in the loss value is due to the absence of 'regularization error' in the `val_loss` which is present in the calculation of `loss` from training values.

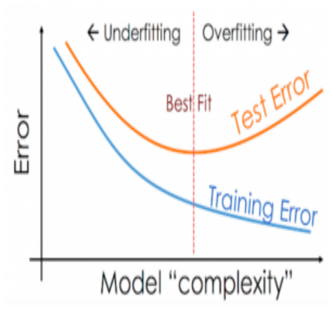
Results

Below is the general view of the relation between the temperature profiles and the rho values. The rho values used are 0.10, 0.36, 0.60, 0.90 and 1.20.



From the plot, we can see the general shape of the temperature curves for rhos that they increase sharply in the first half of the chunk values and reach peaks, after which they decrease and arrive the same temperature value as the left end at the right end. Overall, they show a symmetrical shape with the symmetry at the middle chunk value. This is expected using MP. Furthermore, the curve becomes flatter as the rho values increase, revealing that the temperature of the system with high rho value, i.e., a larger inter-atom distance, is less sensible to the environmental temperature changes.

The methodology discusses above allowed two models to be built that produced very good results, this was tested against the metrics discussed above such as the training and testing mse/losses for the respective model types. Moreover, not only were these metrics recorded and then plotted to see which hyperpareters and architectures provided the lowest scores, but the models were used to predict the temperature profile for values of 'rho' not seen during the models training. These data points have been referenced as `X_test` in the report, the predicted temperature profiles (`y_predict`) were compared to the known actual thermal profiles (`y_test`) which are unseen by the models. The plots for these comparisons of thermal profiles will be shown below for a few testing data points, this will show the models ability to predict on unseen data (known as the flexibility of the model). It was important to be aware of the values calculated for the training and testing losses/mse's as the model could be underfitted or overfitted depending on the scenario observed. The figures below depict these ideas were considered when training the model:



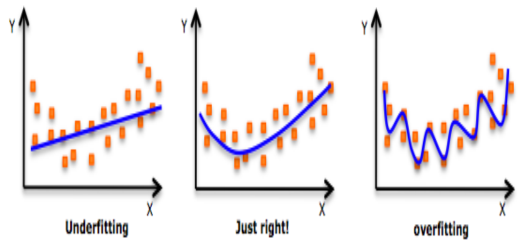


Figure (left) source: Figure (right) source:

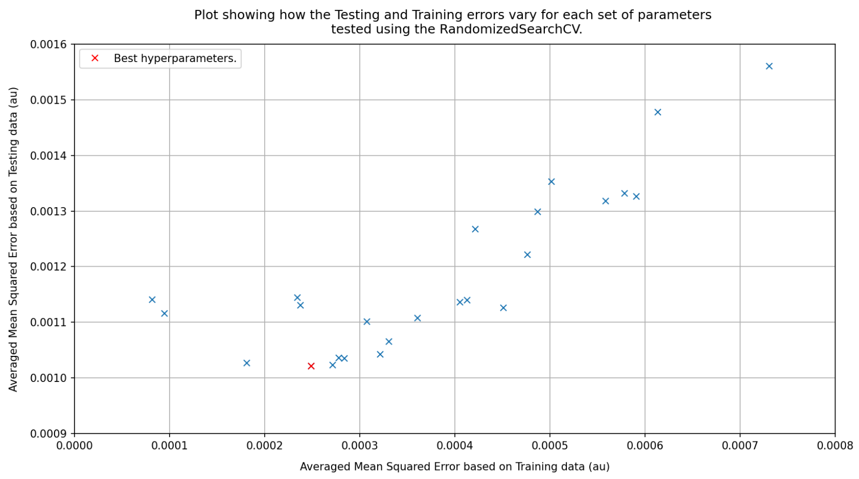
https://vitalflux.com/overfitting-underfitting-concepts-interview-questions/

http://sanghyukchun.github.io/59/

As mentioned previously, this over/under fitting balance was monitered by tracking the `loss` (training data) and `val_loss` (validation data) in the NN scenario, we attempted to minimise both losses while ensuring both values had similar magnitude. If the `val_loss` began to increase significantly above the `loss` metric then the model would have an overfitting issue (and vice versa).

XGBoost

The plot shown below shows the results achieved from a `RandomizedSearchCV` over various parameter ranges for the hyperparameters discussed in the methodology. The plot highlights one of the points in red, which shows the two MSE's for the chosen model (best hyperparameters from the search). It can be observed that the highlighted point doesn't have the smallest training MSE, there are points further to the left of the plot, however these parameter sets have a higher testing MSE which would indicate that the model has overfit the data. An argument could be made to select the model represented by the cross just to the left of the red value due to a similar testing MSE but smaller training MSE however the parameter set with the lowest testing MSE was chosen to be the 'best' model.



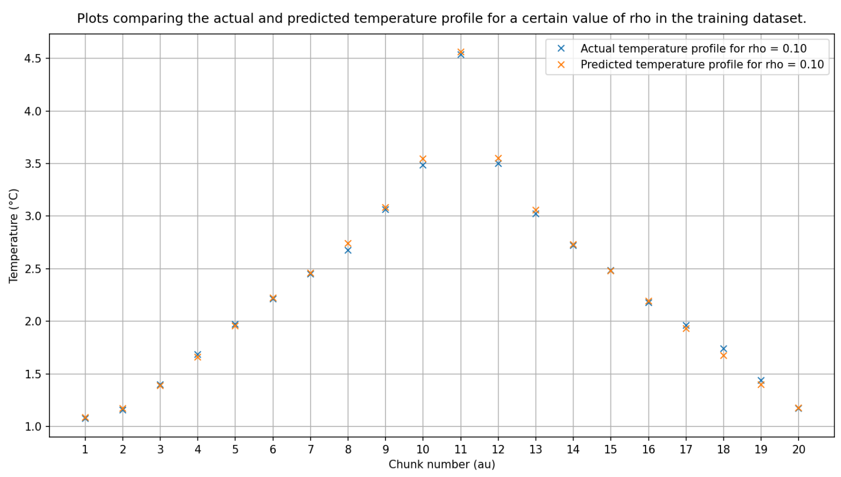
These metrics plottet here are analogous to the `loss` and `val_loss` metrics from the NN scenario, they track the MSE's calculated during the cross valiation process in the XGBoost library.

Another part of the `XGB_first` script produce shows the best model found from the search: `The best parameters for the model are: {'subsample': 0.29482758620689653, 'n_estimators': 808, 'max_depth': 4, 'eta': 0.0146734693877551, 'colsample_bytree': 0.6482758620689655}`. This lists the parameters used for the `XGBRegressor` model that gives the 'best' predictions on the testing dataset. The hyperparameter search performed wasn't completely extensive (did not include all possible hyperparameters) and therefore a better fit to the data may have been achieved. This was implied by the output: `Score for hyperparameter tuning: 0.8498916553721363` out of the possible 1.0 scoring which is a perfect score.

Comparing actual and predicted temperature profiles.

Training data

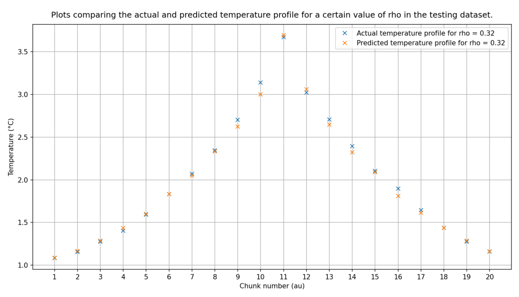
To initially check the accuracy of the model produced it was tested using the training data (`X_train`) which unsurprisingly produced very good results. One of the plots produced for an extreme value of 'rho' (0.1 nm) is shown below, it was seen that all predictions were very close to teh actual profiles recorded.

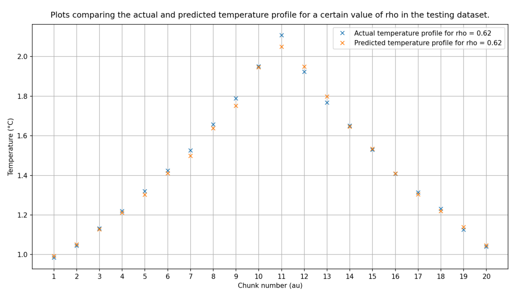


This is not enough to conclude the model can accurately predict the temperature profiles for unseen data, the main part of validating the model is checking the predictions or testing data (`X_test`) which can be seen in the section below.

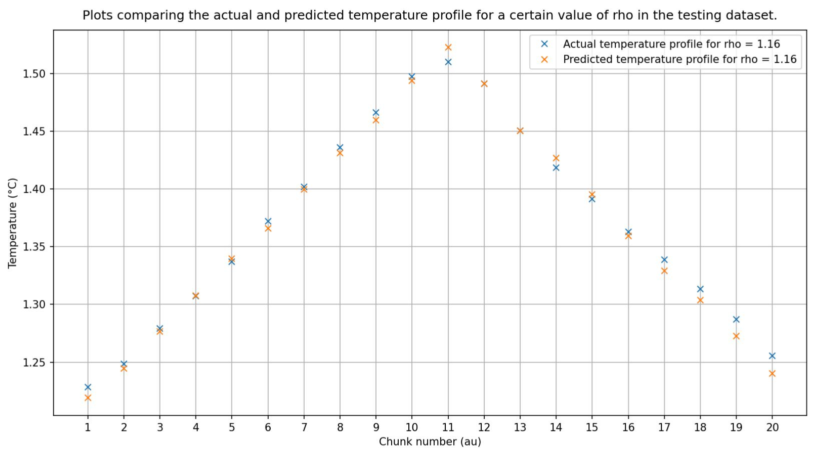
Testing data

To actually evaluate the validity of the model produced in the training process we tested the model on data that was 'unseen' by the network during training, know as the X_test. The model produced (with optimal hyperparameters) was then used with the `predict` function to obtain a predicted temperature profile for various 'rho' values. The plots below show three of the 'rho' values present in the `X_test` data set (for XGBRegressor there were 12 values of rho in this test set) to give an idea of the accuracy of the model produced without an abundance of plots.





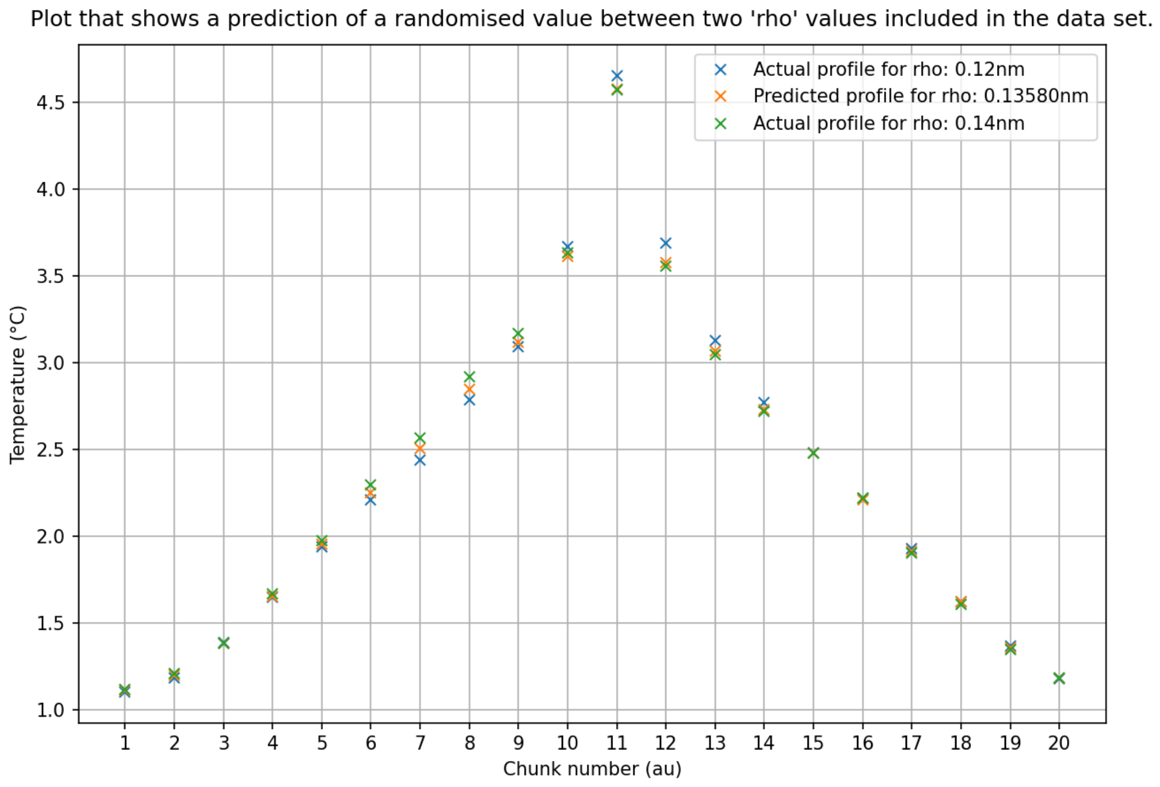
For the first two plots ('rho' = 0.32, 0.62) the trends seen in the figures show that the model fits the data very well for chunk values near the edges of the system (high and low chunk values) where the system is cooler and the model doesnt predict as accurately near the centre (chunk 11) where the temperature values increase more rapidly and reach their peak values. Overall the model seems to predict these temperature profiles relatively well.



It can be seen in the final plot that there is a slightly worse fit observed for 'rho'=1.16 as the fit is no longer as good beyond chunk 11, the model fits very well for chunks 1-8 but then there is a larger error beyond this. This indicates that the `XGBRegressor` model overall predicts thermal profiles more accurately for the lower spectrum of 'rho' values and also has better predictions for less extreme temperature/gradient values (near the centre of the system). This observation can be expected, to improve the models accuracy near the centre of the system it may be beneficial to take temperature measurements more often (increasing the number of chunks near the middle of the system) due to the more extreme changes in temperature observed.

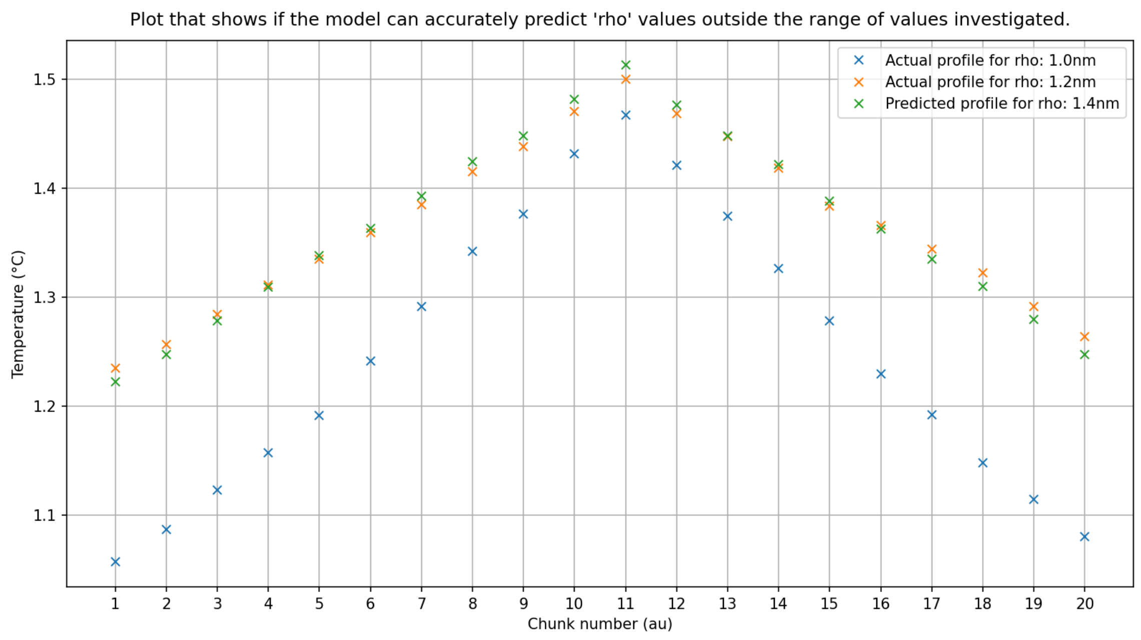
Further checks

Testing a randomised 'rho' value in between the 0.02 steps taken for the rho data points:



This plot shows good accuracy of the model for 'rho' values that do not necessarily have an increment of 0.02 nm, the model 'accurately' predicts the temperature profile of this randomised rho value to be positioned in between the two known profiles. This increases the overall validity of the model found as the predictions are flexible enough to work for values that aren't 'expected' by the models framework.

Testing a value of rho outside the range of 'rho' values investigated:



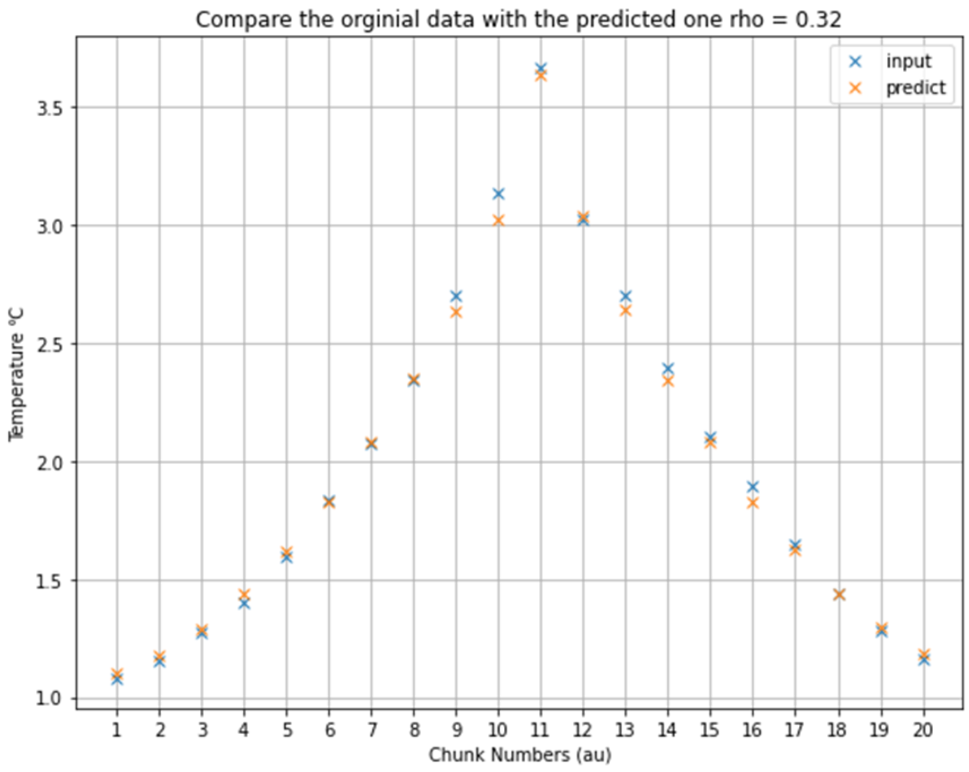
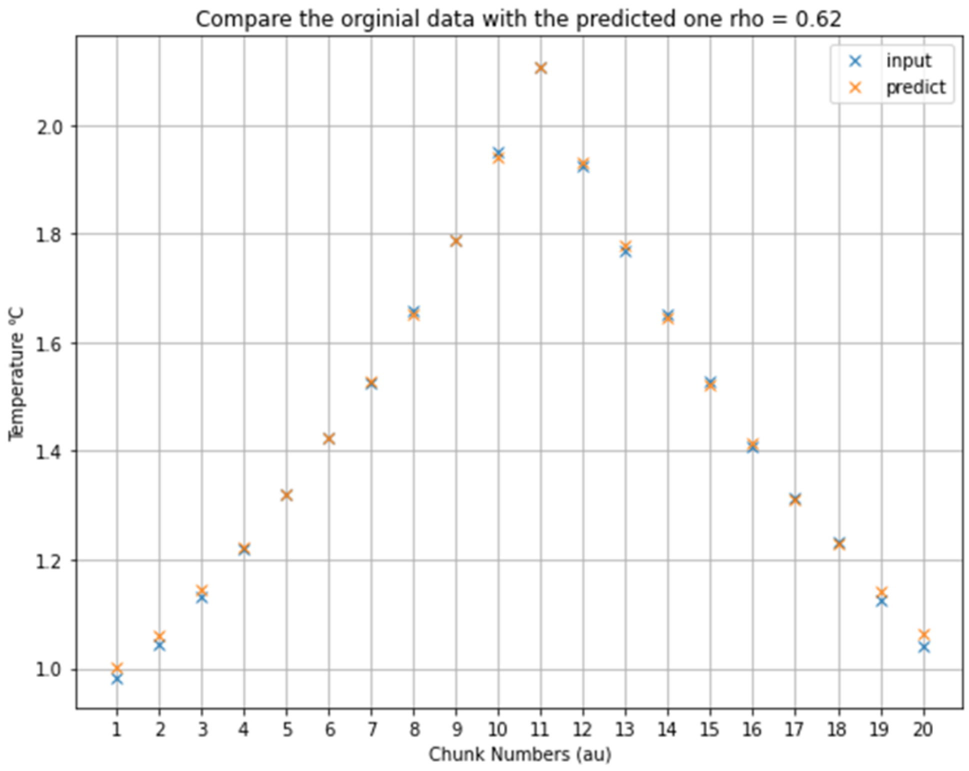
This plot shows that the model fails to predict the temperature profile for a value of 'rho' that is outside the range of values investigated. This is to be expected, if we wish for our model to perform well for a value such as 1.4 nm we must provide the model with data in this vicinity. This could be a further extension to the project, the value of 'rho' is an identifying feature of the material type. In this project, specific material types weren't explicitly considered (only a range of 'rho' values) however when different material types were modelled the value of 'rho' is unlikely to reach these large values (1.0 nm and greater) as this isn't physically realistic for many systems-usually lattice parameters range between 0.3 and 0.8 nm.

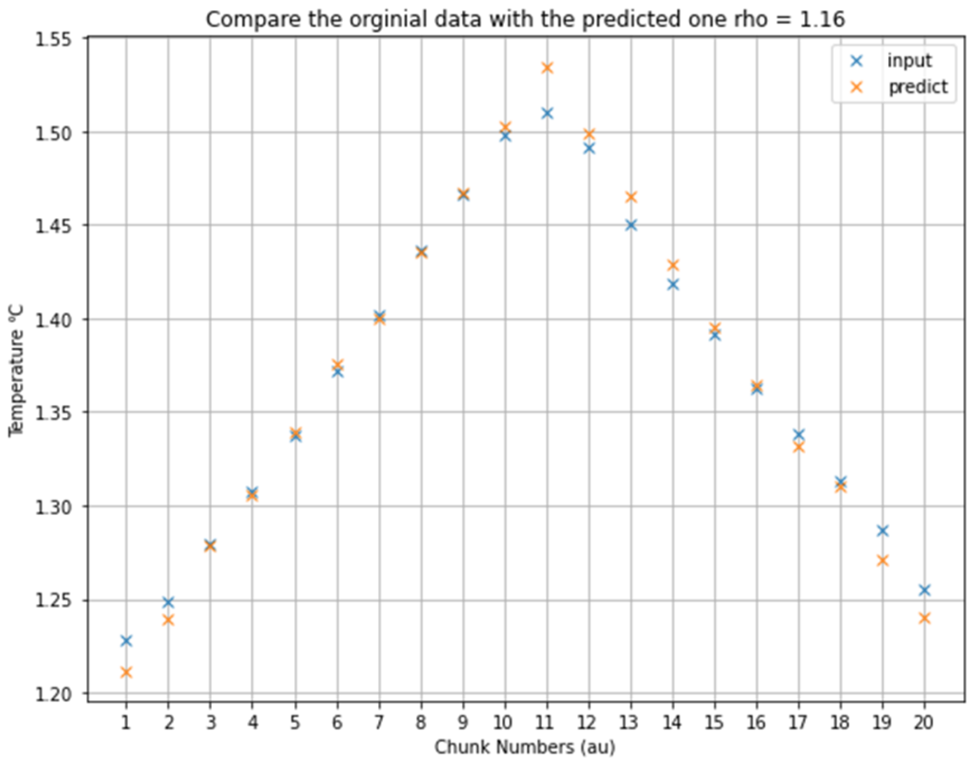
Neural Networks

The NN model consists of one input layer, two Dense layers with ‘relu’ activation and one output layer. The ‘relu’ activation makes sure our predicted temperatures are non-negative. The mean square error of the model is 0.0014417392 with loss as 0.011. These can be further decreased by rerunning through 2.py, after which the mean square error is down to 0.0012669308 with loss as 8.6590e-04. The learning-rate is set to be 0.02.

Comparing actual and predicted temperature profiles.

The testing rho values are taken from the X_test due to the train_test_split function following what we have done similarly for the XGBoost model --- we should test our NN model on the data that was not used for training to see its validity. The three plots below show the comparison between the actual temperatures (the input label) and the predicted ones. To facilitate comparison between two models, the same set of rho values was selected to validate this NN model.

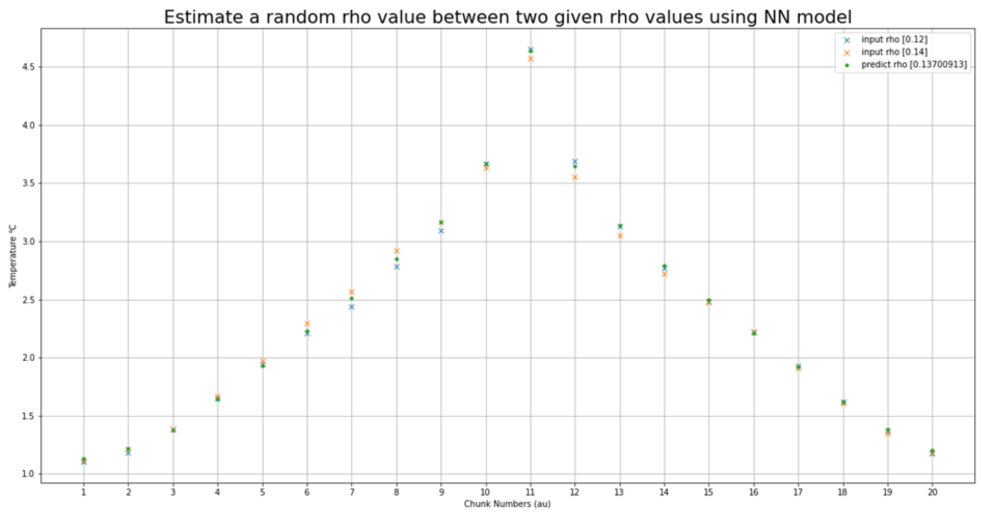

Compare the orginial data with the predicted one rho = 0.32
input
predict
Temperature °C
Chunk Numbers (au)
3.5
3.0
2.5
2.0
1.5
1.0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

Compare the orginial data with the predicted one rho = 0.62
input
predict
Temperature °C
Chunk Numbers (au)
2.0
1.8
1.6
1.4
1.2
1.0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20



From the three plots above, we can see that as the rho values increase, the prediction becomes less accurate from the fact that the ‘input’ and the ‘predict’ points are farther from each other, although, according to the y-axis, this discrepancy is not significant. When rho = 0.32, the difference between the actual temperatures and the prediction is the smallest, while, when rho = 1.16, they diverge from each other. In the case of rho = 1.16, the discrepancy is conspicuous for the extreme chunk numbers while negligible for the rest. However, although the discrepancy exists, the prediction shows a similar curve shape. Considering the overall mean square error and the loss value in the model, the discrepancy is acceptable.

Prediction

This is the test of a randomised rho value between two given rho values. To further examine the performance of the model, we carried a random rho value test that does not exist in the original data set to see how accurate the prediction was. If the model can predict with high accuracy, the predicted temperature positions of the randomly selected rho value should be between the temperature profiles of two adjacent rho values for the same chunk number, which can be viewed as a stepwise increment in rho. Below is the test of a randomised rho value, 0.13726137 between two given rho values, 0.12 and 0.14. It can be observed that most of the points for the random rho value lies between the two given temperature profiles, although there are two outliers (Chunk number: 0 and 20). Thus, we can claim that the prediction corroborates the increasing trend of the temperature curves as the rho values decreases and hence the model is validated.



The plot shows good accuracy of a randomised rho value that does not exist in the original data set from the observation that most of the points lie between the two given temperature profiles.

Conclusions

This project utilized the two machine learning models, Neuron Network and XGBoost Models, to predict the thermal profile of a system from the value of the lattice constant (rho). Training data was obtained from the `overall_final.py` based on an equilibrium system. From the results, both models predict with high accuracy from the comparisons between the actual and the predicted temperature values as well as the randomised rho values test. The current NN model is of two dense layers with 'relu' actvation while it is feasible to modify the parameters such as changing the number of layers and the corresponding activations and varying the `learning_rate` and the epoch value to give a lower mean square error and loss but this will lengthen the computation time.

Bibliography

[1]https://www.rc.ucl.ac.uk/docs/Other_Services/Aristotle/

[2]https://en.wikipedia.org/wiki/Lattice_constant#:~:text=A%20lattice%20constant%20or%20lattice,between%20atoms%20in%20the%20crystal.

[3] https://www.bing.com/search?q=muller+plathe+algorithm+explained&qs=n&form=QBRE&sp=-1&pq=muller+plathe+algorithm+explaine&sc=0-32&sk=&cvid=BF76B4FC1F564E7F896A6DB95EE61DA5&ghsh=0&ghacc=0&ghpl=

[4] [https://towardsdatascience.com/gridsearchcv-or-randomsearchcv-5aa4acf5348c]