Bachelor Thesis

Uncertainty in Graph Neural Networks, applied within Quantum Chemistry

Name:	**Student number:**
Lars Lohmann	s173875

Supervisor:	**Title:**
Mikkel Nørgaard Schmidt	Associate Professor

Department:

Department of Applied Mathematics and Computer Science, Technical University of Denmark

1 Abstract

Contents

2 Acknowledgements

The completion of this project would not have been possible without the dedicated supervisor efforts from Mikkel Nørgaard Schmidt.

Mikkel Nørgaard Schmidts help has manifested through all of the stages of the project, from planning, to scoping, to executing and finalizing. His inviting presence allows for curiosity to an extend very few prior of prior supervisors have managed. His own curiosity towards the topic is contagious. His on the spot questioning and advice, has contributed to the bettering of the project, and a deepening of the knowledge around the subject matter.

When things get tough, his calming manner gave the mental surplus that was needed to finalize the work, and deliver on this project.

I would recommend his counselling efforts to any student at DTU, whether it being in a project or in a teaching capacity.

Thank you!

Lars Lohmann - s173875

3 Introduction

The implications of predicting molecular properties and dynamics have both large industrial- and research implications within the natural- and life-sciences. The large amounts of compute, necessary for simulation, often makes desirable metrics intractable to estimate. The efforts of this study, will help provide tools for industry to reduce lead time and risk in development fields such as drug discovery and material sciences for energy storage[1].

These interests, have for many years been supported by research and development methods within the traditional field of chemistry, with emphasis on estimation techniques, that carry a heavy computational load like traditional molecular dynamics (MD)[2] or Density Functional Theory[1]. Therefore, despite the emergence of cloud-data-centers and power full algorithms available at lower cost to research, some problems still exist which are intractable with known methods.

That fact, has prompted the emergence of the computational fields within Artificial Intelligence (AI), such as Machine Learning (ML), Deep Learning (DL) and Geometric Deep Learning (GDL), to enrich the solution space. These methods produce models which tries to decrease the lead time for predictions, while maintaining an acceptable error-rate[3]. Specifically the field of Geometric Deep Learning, has emerged with a viable set of data-driven methods, for predicting molecular properties and dynamics[4]. Geometric Deep Learning comprises a set of methods, which tries to enrich the initial data structures with a geometric prior. One of these methods is graph neural networks (GNN's) which structures the molecular-data in graphs, which is comprised of nodes and edges between nodes (representing atoms and bonds respectively), in an i.e euclidean space[4]. These models can encode information such as distances, angles or directions between nodes, allowing also for symmetry-priors of the invariant and equivariant types[5]
The most popular architecture within GNN's is the Message Passing Neural Network (MPNN)[4]. This architecture, allows for iterative updates of message information between nodes in the graph representation, allowing for exchanges of information between nodes based on their relationship to each other. Specific models that have shown promise within the chemical space, is the Directional Message Passing Neural Network (DimeNet) model[3], the Multiplex Molecular Graph Neural Network (MXMNet) model [6], and the polarizable atom interaction neural network (PAINN) model[7]. Central to these models is the graph representation of information, allowing for node-features to update based on the information of neighbouring nodes. Where they differ is in i.e their definition of the term neighbour. Traditionally, the concept of a neighbour node, is mainly defined by the euclidean distance between the nodes[8]. This however, DimeNet challenges by including directional information in the definition, while maintaining critical symmetry properties in the model. MXMNet segments node representations into two segments, those representing a covalent influence on the node, and those representing a Wan der Waals influence on the node, allowing them to be modeled seperately and weighed based on importance[6]. In

the case of the PAINN-model, it introduces novel ways to do equivariant operations in cartesian space, allowing the model achitecture to outperform among other the DimeNet model, on numerousprediction tasks.

Since significant risk is embodied in the task of predicting chemical and biochemical dynamics, due to end-uses often being utilized in pharmaceutical products, proper mechanisms for estimating the the uncertainty around prediction methods, and mitigating the impact of high uncertainty through calibration, are needed. Traditionally, data-driven methods within machine learning and deep learning, have problems with providing the necessary uncertainties, on the metrics being predicted, making calibration for more robust overall performance hard[9][1][10]. Providing methods for estimating uncertainty, and calibrating, would allow for defaulting to more traditional methods for estimating chemical properties, when uncertainty is high in the data-driven models[1].

In support of producing such methods, uncertainty, can effectively be distinguished between two concepts, namely epistemic- and aleatoric- uncertainty[1][10]. Aleatoric, also known as statistical uncertainty, refers to the uncertainty inherent in experimental outcomes, due to random effects. This class of uncertainty is classified as irreducible, due to no amount of additional information in the modelling effort, being able to reduce this type of uncertainty[10]. Epistemic, also called systemic uncertainty, refers to the uncertainty in a model, produced by lack of information, and can therefore be reduced. These two elements of uncertainty are considered by researchers to be the sole components of total uncertainty, their sum being equal to total uncertainty in an experiment[10]. Specific methods for estimating these classes of uncertainty in neural networks, have been proposed in recent years. In [11], Three methods for estimating uncertainty for in Deep Neural Networks (DNN's). These three methods are Monte Carlo Dropout (MC Dropout), Ensembling, and Bootstrapping. MC Dropout, relies on training neural networks (NN's), with dropout regularization, and after training producing a set of samples, with different random masks. The uncertainty can then be estimate by looking at the variance over these predictions. Ensembling relies on training the same model architecture, on the same data set with random initializations. The uncertainty can then be estimated by looking at the variance of the predictions of the ensemble of models. Evidence suggests that these ensembles produce better results, when the ensemble is 'diverse'[12]. These recent results indicate implementing methods like early stopping (A method that breaks the training-loop if a validation metric stagnates), and weight decay (A method that penalises large weights in the neural network), will increase this diversity for the better. These methods applied to ensembling, is viewed as a discrete method from ensembling itself[11] called anchored ensembling. Similar to ensembling is bootstrapping, where a set of models are trained like in ensembling, but this time of different subsets of the data set. These subsets are picked with replacement at random, and all subsets will effectively have some overlap in data points, while still having some diversity in points among each other. The conclusion in [11] is that bootstrapping and ensembling are both superior methods to MC Dropout, while being inconclusive on which is the better method

between bootstrapping and ensembling.

The overall goal of this project, is to compare two ensembles of five machine learning models in each, which all are of the type Message Passing Neural Networks with architecture inspired by the PAINN architecture[7]. PAINN has been established as a state of the art method, within the space of GNN's in the field of chemistry, which was essential for the choice of the architecture. The individual models in the ensemble, output a single metric, trying to predict the binding energy of the catalyst process represented by its input. These predictions are then combined to produce a mean prediction and a variance for each ensemble. although the individual models are similar in architecture, they have different hyperparameters in their internal representations of molecules. Specifically the state representation of scalar- and vector-properties of the input graphs. One ensemble of models will have a half (64) of the dimensions-size for representing both scalar- and vector-properties, compared to the other ensemble (128). The individual models in the ensembles will be trained with weight decay and early stopping, in alignment with current state of the art methods, in order to produce diversity.

The models will be trained on a molecular data set[13], comprising of 7.053 molecules. This dataset contain atom-types, coordinates, and binding energy in kcal per mol, where the latter is our regression target. The main contribution of this paper, will be to assess the influence of this hyperparameter difference on modelling performance measured by the mean squared error (MSE) between the predictions of the ensembles and the actual target values, and epistemic uncertainty founded in the variance in the predictions of the ensembles inspired by[14], and [15]. The Expected Normalized Calibration Error (ENCE), will also be calculated as a supplementary measure.

Specifically, this paper asks:

Does the size of internal state representations, impact prediction error and uncertainty in ensembles of Message Passing Neural Networks

The goal of this project has been to implement the model architecture from scratch, utilizing tools and packages from the *PyTorch* library. This with the intention of achieving a more detailed understanding of the modelling task, uncertainty in modelling, and the model structure itself. This specifically means implementing the MPNN architecture[7], and the ensembling procedure from scratch. Another aim of the project, is to implement the model architecture, and produce viable results for the purpose of answering the above question, not producing state of the art results, viable for comparison with other benchmarks. The comparison between the two ensembles will be based be founded in the metrics mentioned prior, but without the statistical rigor and testing, that are some times relevant for these studies. The comparison will be done in a more qualitative manner. The project is limited in terms of providing a measure for both epistemic and aleatoric uncertainty. Since the individual models in the ensembles only predict a single value, trying to minimize the MSE-loss function, and not an output mean and

variance like done in Busk2021[1], and proposed in [14], aleatoric uncertainty can not be estimated. The distinquishing between the two types of uncertainty, while being the state of the art for evaluating uncertainty in modelling tasks, is not attainable due to the scope of the project. The choice of estimating uncertainty via the variance and mean over ensemble predictions, are viable options performed in Tran[15], and proposed by Scalia[11], but critiqued for its lack of capabilities in Lakshminarayanan[14]. The sole data set in scope of this project is the C-c catalyst dataset[13]. The choice came down to the accesability of the data set, being limited to 7.053 molecules, with easy to comprehend data structures. This choice supported the desire to emphasize understanding the modelling task and uncertainty in modelling.

The following sections of this paper, namely[3,4], we will go through relevant concepts and theories, necessary for assessing the results and methods in this paper. This is followed by a description of the experimental methods, and a presentation of the data set, on which the methods have been utilized in sections[5,6]. Lastly, results followed by a conclusion and a discussion of the potential points of error, which influence the outcomes of the experiment in sections[7,8,9].

4 Theory

In this section, we will discuss the theoretical foundations of message passing neural networks (MPNNs) and ensembles for predicting the binding energy of molecules in a catalyst process. The section will start with a note on symmetry, specifically detailing the usage of invariant and equivariant representations in the model. Moving on to a detailing of the modelling architecture, ending in a presentation of the ensemble structure and metrics used in the experiment.

4.1 Symmetry

A large part of the contribution from the PAINN paper[7]. is the emphasis on building and modelling equivariant features. The interest in equivariance, comes from the ability to express more information about a graph structure, at a lower level of computational cost. This ties in with the search for modelling approaches that descrease lead-time on predictions, while maintaining an acceptable error-rate. let $\vec{\mathbf{x}}$ be a feature, and o denote an operation. $\vec{\mathbf{x}}$ is a rotational equivariant feature, under the operation o, if the below equation is satisfied for any rotation matrix $R \in \mathbb{R}^{3\times3}$1:

$$R\vec{\mathbf{o}}(\vec{\mathbf{x}}) = \vec{\mathbf{o}}(R\vec{\mathbf{x}}) \tag{1}$$

And $\vec{\mathbf{x}}$ is a rotational invariant feature, under the operation o, if the below equation is satisfied for any rotation matrix $R \in \mathbb{R}^{3\times3}$2:

$$\mathbf{o}(\vec{\mathbf{x}}) = \mathbf{o}(R\vec{\mathbf{x}}) \tag{2}$$

In order to detail whether any of these equations hold, one needs to consider which operations, make this the statements true for directional information. The paper of inspiration [7] highlights a list of operations, that equivariant MPNN's in particular can use in order to preserve equivariant information properties on their features:

- Any (nonlinear) function of scalars: $\mathbf{f}(\mathbf{s})$
- Scaling of vectors $\mathbf{s} \circ \vec{\mathbf{v}}$
- Linear combinations of equivariant vectors: $\mathbf{W}\vec{\mathbf{v}}$
- Scalar products: $\mathbf{s} = \|\vec{\mathbf{v}}\|^2, s = \langle \vec{\mathbf{v}}_1, \vec{\mathbf{v}}_2 \rangle$
- Vector products: $\vec{\mathbf{v}}_1 \times \vec{\mathbf{v}}_2$

This list entails a linearity constraint on the operations on equivariant features[7]. If the linearity constraint is not satisfied, then the feature will not be equivariant, and directional information will not be preserved.
The subsequent section presents a number of features (presented also as representations below) and operations relevant to the MPNN. These need not all be equivariant, invariant, linear or non-linear. What constitutes an equivariant MPNN in the eyes of Schütt et. al.**??**, is not that all operations on directional information are linear, and equivariance is preserved globally. As long

as an equivariant representation containing directional information, is obeying the constraints of the list above and thereby equation1, directional information is preserved, and can be allowed to diffuse through the model representations.

4.2 Message Passing Neural Networks

The key idea behind MPNNs is the concept of message passing, where each node in the graph exchanges information with its neighbors in an iterative manner.
Let $G = (N, E)$ be a graph, where N is the set of nodes and E is the set of edges. Each node $n \in N$ is associated with an invariant representation vector $\mathbf{S}_n \in \mathbb{R}^{Fx1}$, an equivariant representation vector $\vec{\mathbf{v}}_n \in \mathbb{R}^{Fx3}$ and a position in 3D space $\vec{r}_i \in \mathbb{R}^3$. F denotes the number of features in the state representation (in our case, $F = 64$ and $F = 128$). Each edge $(i, j) \in E$ is associated with vector difference $\vec{r}_{ij} = \vec{r}_i - \vec{r}_j$. A node $n_j \in N$ is said to be a neighbour of a node $n_i \in N \setminus \{n_j\} = \mathscr{N}(i)$ if n_j is within the cutoff distance of n_i: $\mathscr{N}(i) = \{j | \|\vec{r}_{ij}\| \leq r_{cut}\}$.
The message passing process in an MPNN can be described by the following equations 3 and 4 following the notation of [7].

$$\vec{\mathbf{m}}_i^{v,t+1} = \sum_{j \in \mathscr{N}(i)} \vec{M}_t(\mathbf{S}_i^t, \mathbf{S}_j^t, \vec{\mathbf{v}}_i^t, \vec{\mathbf{v}}_j^t, \vec{r}_{ij}) \tag{3}$$

$$\vec{\mathbf{s}}_i^{v,t+1} = \vec{U}_t(\mathbf{S}_i^t, \vec{\mathbf{v}}_i^t, \vec{\mathbf{m}}_i^{v,t+1}) \tag{4}$$

The U and M functions respectively being the update, and message functions, taking into account scalar, vector and directional features. These equations utilize both invariant and equivariant representations, letting representations interact, in accordance with knowledge surrounding the chemical descriptor trying to be predicted. The underlying operations of the functions U and M, have a linearity constraint on equivariant representations of directional information, in order to maintain information, throughout the modelling process[4]. The message and update functions will be designed, such that scalars $\mathbf{s}_i$ and vectors $\vec{\mathbf{v}}_i^t$ are updated in an iterative manner via the output residuals $(\Delta \mathbf{s}_i^t, \Delta \vec{\mathbf{v}}_i^t)$ produced by the message and update functions. this entails that for the message block output is:

$$\sum_{j \in \mathscr{N}(i)} \vec{M}_t(\mathbf{S}_i^t, \mathbf{S}_j^t, \vec{\mathbf{v}}_i^t, \vec{\mathbf{v}}_j^t, \vec{r}_{ij}) = (\Delta \mathbf{s}_i^m, \Delta \vec{\mathbf{v}}_i^m) \tag{5}$$

And for the update block output is:

$$\vec{U}_t(\mathbf{S}_i^t, \vec{\mathbf{v}}_i^t, \vec{\mathbf{m}}_i^{v,t+1}) = (\Delta \mathbf{s}_i^u, \Delta \vec{\mathbf{v}}_i^u) \tag{6}$$

These residuals will be further developed in the sections coming.

4.2.1 Message Block

Further, we define the residual update of invariant scalar7-representations in the message block, based on the expression derived earlier in equation5:

$$\Delta \mathbf{s}_i^m = (\phi_s(\mathbf{s}) * \mathscr{W}_s)_i = \sum_j \phi_s(\mathbf{s}) \circ \mathscr{W}_s \left(\left\| r_{ij} \right\| \right) \tag{7}$$

Worht mentioning about the above expression, are that the index 's' refers to a certain split of the $\phi \circ W$-product seen in figure1. All indexes of 's', 'v', or a combination of the two in the below equations, will be detailing what part of a split it belongs to. This comes down to ϕ and $\mathscr{W}$ being shared networks, that are re-used for multiple parts of the modelling. Following the above equation7, two expression are utilized, namely ϕ_s and $\mathscr{W}_s$. ϕ_s represents atomwwise layers, which have undergone transformations through two linear layers and a SiLU-function, introducing some non-linearity potentially for better gradient flow, via the smooth gradient. Conceptually it can be interpreted as an expansion of the embedded atomwise neighbours of a node. Let $f(x)$ denote a linear layer:

$$f(x) = w \cdot x + b \tag{8}$$

and $SiLU(x)$ denote the SiLU-function:

$$SiLU(x) = x \cdot \left(\frac{1}{(1 + e^{-x}))} \right) \tag{9}$$

Then ϕ is defined as:

$$\phi_s(\mathbf{s}) = f_1(SiLU(f_2(\mathbf{s}))) \tag{10}$$

Worth noting about the above equation10, are the indexes of the linear layers. These are maintained to emphasize, the individuality of their parameters throughout the whole modelling process.

The rotationally invariant filters $\mathscr{W}_s$ are defined as, are linear combinations of a radial basis function (RBF)11[4]. The radial basis function outputs are chosen such that it is centered around twenty points, $C = \{1, 2, 3, \ldots, 20\}$. These twenty points are centers of the filter, and their units are Angstrom. These centers are chosen, such that they cover all distances in the data set, which is confirmed in a later section8. The expansions provide a representation of similarity or dissimilarity, between the input relative position $\vec{r}_{ij}$, and the chosen points C [16]. The function takes directional information $\|\vec{r}_{ij}\|$ and a cutoff r_{cut}. The rbf acts as filter generating function[17], effectively quantizing the directional information. The function is defined as:

$$\mathbf{RBF}(\|\vec{r}_{ij}\|) = \sin\left(\frac{C\pi}{r_{cut}} \cdot \|\vec{r}_{ij}\| \right) / \|\vec{r}_{ij}\| \tag{11}$$

This radial basis is then expanded on by a linear layer f, before the cosine cutoff function is applied. This effectively means that, that atoms beyond the cutoff radius, does not contribute to the representation[2]. The cosine cutoff function is defined as:

$$\mathbf{f}_{cut}(\|\vec{r}_{ij}\|) = \begin{cases} 0.5 \cdot \cos\left(\frac{\pi \|\vec{r}_{ij}\|}{r_{cut}} + 1 \right) \cdot f(\mathbf{RBF}(\|\vec{r}_{ij}\|)) & \text{if } d \leq r_{cut} \\ 0 & \text{if } d > r_{cut} \end{cases} \tag{12}$$

Worth mentioning is that the cosine cutoff implemented in this project, has an added factor compared to Behler2011[2], namely $f(\mathbf{RBF}(\|\vec{r}_{ij}\|))$. This is deemed missing by the project, and is therefore added. The full representation of the rotationally invariant filters are therefore:

$$\mathscr{W}_s = \mathbf{f}_{cut}(\|\vec{r}_{ij}\|, f_3(\mathbf{RBF}(\|\vec{r}_{ij}\|))) \tag{13}$$

Next also utilizing continous-filter convolutions $\mathscr{W}$, we define the residual equivariant vector update, following the definition in 5, from the message block as:

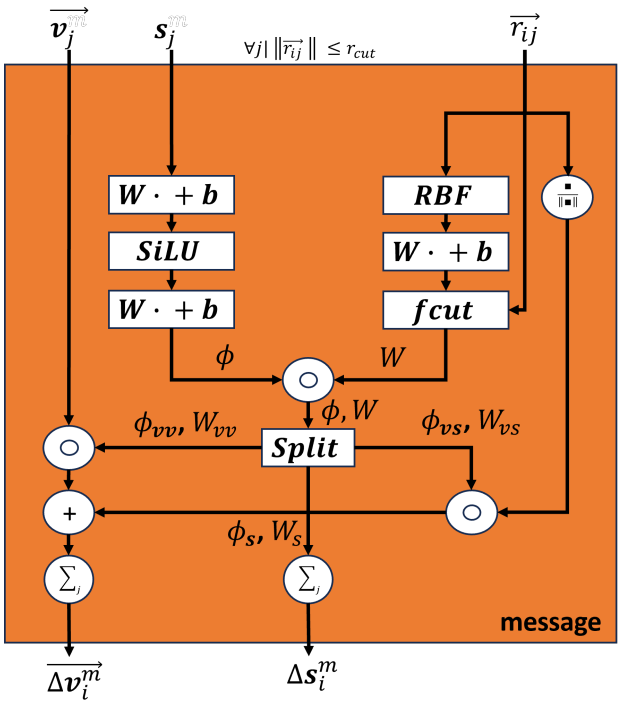
$$\Delta\vec{\mathbf{v}}_i^m = \sum_j \vec{\mathbf{v}}_j \circ \phi_{vv}(\mathbf{s}_j) \circ \mathscr{W}_{vv}\left(\left\|\vec{\mathbf{r}}_{ij}\right\|\right) + \sum_j \phi_{vs}(\mathbf{s}_j) \circ \mathscr{W}'_{vs}\left(\left\|\vec{\mathbf{r}}_{ij}\right\|\right) \frac{\vec{\mathbf{r}}_{ij}}{\left\|\vec{\mathbf{r}}_{ij}\right\|} \tag{14}$$

The equation14 consists of two terms. The first being a convolution of an invariant filter $\phi_{vv}(\mathbf{s}_j)$ being scaled/weighted with equivariant features $\sum_j \vec{\mathbf{v}}_j \circ \phi_{vv}(\mathbf{s}_j)$[7] as well as the rotationally invariant filters $\sqcup\rceil\S\sqcup\mathscr{W}_{vv}(\|\vec{r}_{ij}\|)$. This term propagates prior directional information, from neighbour nodes n_j to the node n_i, stored in the equivariant representation vectors $\vec{\mathbf{v}}_j$, after the initial message passing round since there is no directional information in the initialization of the vector. The second term is a convolution of invariant features $\phi_{vs}(\mathbf{s}_j)$ with an equivariant filter $W'_{vs}\left(\left\|\vec{\mathbf{r}}_{ij}\right\|\right)\frac{\vec{\mathbf{r}}_{ij}}{\left\|\vec{\mathbf{r}}_{ij}\right\|}$ Which is the gradient of the invariant filter:

$$\nabla\mathscr{W}_{vs}(\|\vec{r}_{ij}\|) = \mathscr{W}'_{vs}(\|\vec{r}_{ij}\|)\frac{\vec{\mathbf{r}}_{ij}}{\left\|\vec{\mathbf{r}}_{ij}\right\|} \tag{15}$$

The gradient part of the expression $\mathscr{W}'_{vs}(\|\vec{r}_{ij}\|)$, also being an invariant filter, is modeled directly, without taking the derivative[7]. This concludes the equations related to the message block. A full architectural overview of the block, can be seen in the figure below1:

Figure 1: Message Block



4.2.2 Update Block

The update block builds on the information gained from the message block. It takes each individual node $n_i = (\mathbf{s}_i, \vec{\mathbf{v}}_i)$ And runs ite through a set of transformations, before arriving at yet another set of residual update of the scalar and vector representations respectively.
From the definition of an update function made in equation6, the residual update of the invariant scalar properties are defined for the update block as16:

$$\Delta \mathbf{s}_i^u = \mathbf{a}_{ss}\left(\mathbf{s}_i, \|\mathbf{V}\vec{\mathbf{v}}_i\|\right) + \mathbf{a}_{sv}\left(\mathbf{s}_i, \|\mathbf{V}\vec{\mathbf{v}}_i\|\right) \langle \mathbf{U}\vec{\mathbf{v}}_i, \mathbf{V}\vec{\mathbf{v}}_i \rangle \tag{16}$$

In general the update function, utilizes a shared network a, just as the message function did with ϕ and $\mathscr{W}$. This network is defined as by inputs of the invariant scalar representation $\mathbf{s}_i$ and the normalized linear combination of the equivariant vector representation $\|\mathbf{V}\vec{\mathbf{v}}_i\|$. both U and V are learned structures by the network, and are applied inorder to produce the linear combinations of equivariant vector representations with dimensions: $V \in \mathbb{R}^{F \times F}$ and $U \in \mathbb{R}^{F \times F}$. The network applies two linear transformations to the input f, as well a SiLU transformation $SiLU$. The full expression of a can be seen in the following equation:

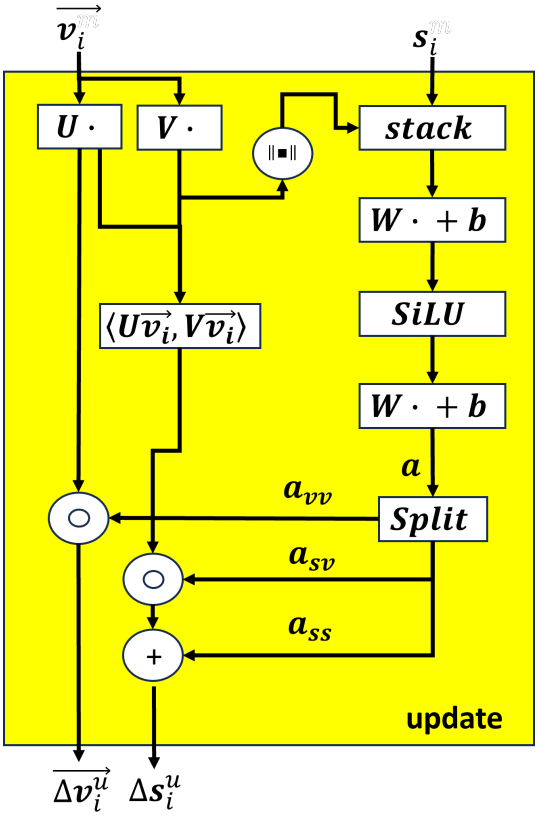
$$\mathbf{a}(\mathbf{s}_i, \|\mathbf{V}\vec{\mathbf{v}}_i\|) = f_4(SiLU(f_5(\mathbf{s}_i, \|\mathbf{V}\vec{\mathbf{v}}_i\|))) \tag{17}$$

The residual update of the scalar representations in the update block16 contains two terms. The first term in16 relates to a non-linear scalar representation $\mathbf{a}_{ss}\left(\mathbf{s}_i, \|\mathbf{V}\vec{\mathbf{v}}_i\|\right)$. The second applies the scalar product of the two matrices U and V, as further scaling to the non-linear of the invariant features $\mathbf{s_i}$, as well as the coupling of the scalar representations with the normalized linear combination of the equivariant vector representation similar to the first term. This whole expression imposes some further non-linearity on the scalar representations, as well as letting the equivariant representation influence the output, via the scalar product, and the normalized linear combination.
The residual update of the vector representations in the update block is defined as18:

$$\Delta \mathbf{v}_i^u = \mathbf{a}_{vv}\left(\mathbf{s}_i, \|\mathbf{V}\vec{\mathbf{v}}_i\|\right) \mathbf{U}\vec{\mathbf{v}}_i \tag{18}$$

Which is similarly non-linear scaling $\mathbf{a}_{vv}\left(\mathbf{s}_i, \|\mathbf{V}\vec{\mathbf{v}}_i\|\right)$ of the linear combinations of the equivariant representations, although the combinations are not normalized.
This concludes the section on the update block, a figure of the update-blocks architecture, can be found below2.

Figure 2: Update block



4.2.3 A note on Equivariance

As the initial symmetry section detailed4.1, there are a number of operations which are valid for equivariant representations of directional information. This section will take a short look back on the structures detailed above in the message-4.2.1 and update-block4.2.2 sections, how exactly directional information propagates in the network.

Directional information is propagated from vector differences $\vec{\mathbf{r}}_{ij}$ as input, into many of the model areas. The main storage unit of directional information, is the equivariant vector representation $\vec{\mathbf{v}}_i$. This representation allows for directional information to be stored across iterations, and it is therefore important that its equivariance characteristics are preserved.

If we take a look at the full set of operations applied to the vector representation, we can check for whether equivariance is preserved, by checking the list presented in the section4.1. The operations applied to the vector representation are as follows:

$$\Delta\vec{\mathbf{v}}_i = \Delta\vec{\mathbf{v}}_i^m + \Delta\vec{\mathbf{v}}_i^u = \sum_j \vec{\mathbf{v}}_j \circ \phi_{vv}(\mathbf{s}_j) \circ \mathscr{W}_{vv}\left(\left\|\vec{\mathbf{r}}_{ij}\right\|\right) + \sum_j \phi_{vs}(\mathbf{s}_j) \circ \mathscr{W}'_{vs}\left(\left\|\vec{\mathbf{r}}_{ij}\right\|\right) \frac{\vec{\mathbf{r}}_{ij}}{\left\|\vec{\mathbf{r}}_{ij}\right\|} + \mathbf{a}_{vv}\left(\mathbf{s}_i, \|\mathbf{V}\vec{\mathbf{v}}_i\|\right) \mathbf{U}\vec{\mathbf{v}}_i \tag{19}$$

we further break the expression down, into its seperate terms, we can analyse them one at a time:

$$\sum_j \vec{\mathbf{v}}_j \circ \phi_{vv}(\mathbf{s}_j) \circ \mathscr{W}_{vv}\left(\left\|\vec{\mathbf{r}}_{ij}\right\|\right) \tag{20}$$

$$\sum_j \phi_{vs}(\mathbf{s}_j) \circ \mathscr{W}'_{vs}\left(\left\|\vec{\mathbf{r}}_{ij}\right\|\right) \frac{\vec{\mathbf{r}}_{ij}}{\left\|\vec{\mathbf{r}}_{ij}\right\|} \tag{21}$$

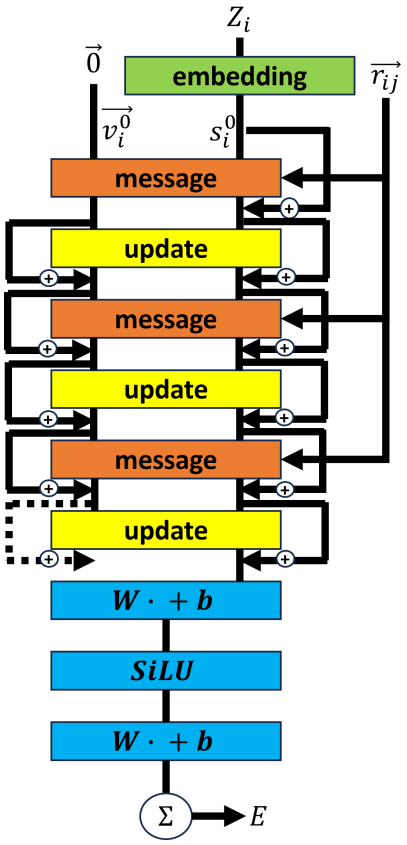
$$\mathbf{a}_{vv}\left(\mathbf{s}_i, \|\mathbf{V}\vec{\mathbf{v}}_i\|\right) \mathbf{U}\vec{\mathbf{v}}_i \tag{22}$$

The first term20 scales the equivariant vector representation $\vec{\mathbf{v}}_i$, with a non-linear term, namely $\phi_{vv}(\mathbf{s}_j)$, and then further scales it with the rotationally invariant filter $\mathscr{W}_{vv}(\left\|\vec{\mathbf{r}}_{ij}\right\|)$. These two operations are both listed in the section4.1. Further, the summing over j, preserves the linearity, as long as the individual operations are linear. Thus, equivariance is preserved. In the second term21, similar operations are applied to the unit vector produced from normalizing the vector differences $\frac{\vec{\mathbf{r}}_{ij}}{\left\|\vec{\mathbf{r}}_{ij}\right\|}$. Two scalings of the vector, one non-linear, the other by an invariant filter. Worth mentioning about the second term, is that it is a source of equivariant directional information. The equivariant representation is initialized, without directional information. The other terms either rely on directional information from the equivariant representation vector or apply non-linear operations to directional information. The last term22, which is a non-linear scaling of linear-combinations of the equivariant representation vector $\vec{\mathbf{v}}_i$. All operations listed in the above section, approved for preserving equivariance.

4.2.4 Full Architecture

The message- and update-blocks are coupled, and stacked, for input to pass through the blocks in a sequential manner. This sequence can be seen in the figure below: 3

Figure 3: MPNN Full Architecture



Worth noting about the full architecture, is the number of coupled message- and update-blocks, which is three. The output of these three coupled sections will take the invariant scalar representations as input for a final set of layers, similar to the ϕ-structure presented earlier. Denote the scalar representations $\mathbf{s}_i \in \mathbb{R}^{F\times 1}$, the feature dimension F, utilizing the linear-layer-8 and SiLU9-expressions from earlier. The expression of the output layer O is then:

$$\hat{\mathbf{y}} = \sum_F O(f_6(SiLU(f_7(\mathbf{s})))) \tag{23}$$

Which produces a set of predicted energies, which will be the output of the model. These output energies $\hat{\mathbf{y}}$, will be compared to the targets of the model $\mathbf{y}$ to calculate the loss, via the mean squared error:

$$L = \frac{1}{N}\sum_{i=1}^{N}(\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \tag{24}$$

This concludes the theoretical walkthrough of the inner workings of the implemented MPNN. The next section will detail the combination of individual model architectures, into a bigger whole, namely an ensemble.

4.3 Ensembles

Ensembles (or Ensembling) is a technique for combining the predictive performance of individual modelling approaches and to as an extension measuring uncertainty in the model. For the purpose of this study, only epistemic uncertainty, can be measured in the models.
In the context of MPNNs, ensembles can be created by training multiple models, in our case, on the same data set, with different initializations, and combining their predictions to produce a mean and a variance among other metrics.

4.3.1 Descriptive statistics of the Ensemble

Let $\mathbf{y}_i$ be the target binding energy for molecule i. Denote an ensemble of T models where t denotes a single model in the ensemble, where $t \in \{1,2,\ldots,T\}$. The ensemble prediction $\tilde{\mathbf{y}}_i$ can be calculated as the mean of the individual model predictions $\mathbf{y}_{i,t}$, as done in[15], and proposed in [11]:

$$\tilde{\mathbf{y}}_i = \frac{1}{T}\sum_{t=1}^{T}\mathbf{y}_{i,t} \tag{25}$$

Further, in order to estimate the epistemic uncertainty in the ensemble predictions, we can calculate the variance based on individual model predictions[15]:

$$\text{Var}(\mathbf{y}_i) = \frac{1}{T}\sum_{t=1}^{T}(\mathbf{y}_{i,t} - \tilde{\mathbf{y}}_i)^2 \tag{26}$$

4.3.2 Uncertainty estimation

Finally as the foundation of estimating uncertainty in the ensemble predictions, we can estimate the expected normal calibration error (ENCE). This method relies on sorting metrics, in K equal sized bins, where $k \in \{1,2,\ldots,K\}$ denotes a single bin. A formal defintion of a bin is given below:

$$bin(x) = \begin{cases} 1, & \text{if } x \text{ falls within the bin range} \\ 0, & \text{otherwise} \end{cases} \tag{27}$$

Effectively outputting a result within a binary range, whether a given value, is within the bin range 1, or outside the bin range 0.

Further, we need to define a bin range. The bin ranges chosen are based on quantiles, which effectively will produce K bins, of equal size.

Given a set of ascendingly sorted numbers $X = \{x_1, x_2, ..., x_n\}$ with n elements. Denote a set of quantiles $P = \{p_0, p_1, ..., p_K\}$ with K elements. These quantiles are fractions of the number of elements in X namely n and the index k of the element in P, specifically $\frac{p_k}{n}$. The p_k-quantile of X, denoted as $Q(p_k)$, is defined as follows:

- Calculate the index of the quantile $q = p_k \cdot (n-1)$.
- If q is an integer, $Q(p_k)$ is equal to the q-th element of X, namely x_q.
- If q is not an integer, $Q(p_k)$ is equal to $\frac{1}{2} \cdot (x_{\lfloor q \rfloor} + x_{\lceil q \rceil})$

In the above, $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ denotes the ceiling and flooring functions respectively. The ceiling function, denoted as $\lceil x \rceil$, rounds a real number x up to the nearest integer greater than or equal to x. The flooring function on the opposite, denoted as $\lfloor x \rfloor$, rounds a real number x down to the nearest integer less than or equal to x.

These quantiles are then used to calculate the bin ranges:

$$binrange(k) = [Q(p_k); Q(p_{k+1})] \tag{28}$$

So for the final definition of the bin, utlizing the quantiles to produce bin ranges.

$$bin_k(x) = \begin{cases} 1, & \text{if } x \in binrange(k) \\ 0, & \text{otherwise} \end{cases} \tag{29}$$

Utilizing the above definition of a bin, we now want to sort all errors made by the individual models $e = y_i - y_{i,t}$ in ascending fashion. We now define $K = \{1,2,\ldots,K\}$ bins, and $P = \{p_0, p_1, ..., p_K\}$ quantiles for the error metric to be sorted into bins. the sorting is done via the below expression for all $k \in \{1,2,\ldots,K\}$ and all models $t \in \{1,2,\ldots,T\}$:

$$bin_k(e) = \begin{cases} 1, & \text{if } e \in binrange(k) \\ 0, & \text{otherwise} \end{cases} \tag{30}$$

We now wish to calculate the root mean squared error (RMSE) on each bin. The RMSE metric, describes the mean of empirical error, made by all the models in the given bin. Denote the number of elements in the bin as N, where the expression of the RMSE, is as follows:

$$RMSE_k = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(e_n^{(k)}\right)^2} \tag{31}$$

We now go through a similar process, for variances over individual predictions and the means of the predictions $\text{Var}(\mathbf{y}_i) = \frac{1}{T}\sum_{t=1}^{T}(\mathbf{y}_{i,t} - \tilde{\mathbf{y}}_i)^2$. We sort them in ascending order. Define $K = \{1,2,\ldots,K\}$ bins, and $P = \{p_0, p_1, ..., p_K\}$ quantiles for the difference metric to be sorted into. the sorting is done via the below expression for all $k \in \{1,2,\ldots,K\}$ and all models $t \in \{1,2,\ldots,T\}$:

$$bin_k(\text{Var}(\mathbf{y}_i)) = \begin{cases} 1, & \text{if Var}(\mathbf{y}_i) \in binrange(k) \\ 0, & \text{otherwise} \end{cases} \tag{32}$$

Now wanting to produce the root mean variance (RMV) on each bin. The RMV metric, describes the variance on predictions, made by all the models in the given bin. Denote the number of elements in the bin as N, where the expression of the RMV, is as follows:

$$RMV_k = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\text{Var}(\mathbf{y}_i)_n^{(k)}} \tag{33}$$

We now wish to produce the ENCE-metric34[1]. This metric is produced based on the RMSE and RMV metrics, calculated further above. These metrics are calculated for each k-bin, for the chosen number of bins K. The ENCE metric is defined below, followed by the RMSE and the RMV.

$$ENCE = \frac{1}{K}\sum_{k=1}^{K}\frac{|RMV_k - RMSE_k|}{RMV_k} \tag{34}$$

These methods are heavily inspired by the methods in the papers[15] and [1]. Elements and approaches have been drawn from both, effectively doing cherrypicking for the scope of this project. The approach by Busk2021, is on models with two output metrics, compared to this projects one. The paper by Tran2019, utilizes a similar approach to describing epistemic uncertainty in the ensemble predictions, made by models with one output metric. But tran2019, lacks in the descriptive metrics that Busk2021 has, i.e RMSE, RMV, and ENCE.

5 Methods

This section will detail the implementation and necessary technological foundation, for applying the theories described, in order to produce the experimental results for answering the research question. The section will also detail the necessary information, for reproducing and understading the experimental results and their becoming.

5.1 Experimental Setup

The experimental setup are comprised of the following components:

- Dataset
- Model
- Training- and Validation-Loop
- Test loop

These four components all together make up the main functionality of the experiment, and will be detailed individually below. The detailing will have its focus on how concepts and theories previously explained, are formatted and shaped, in order to be viable for practical modelling purposes.

5.2 Dataset

The data set is comprised of one row for each node for 695 molecules, with a the following fields for each node:

- Atom-type
- Overall binding energy (target property)
- x-coordinate
- y-coordinate
- z-coordinate
- Graph id

The atom type signifies the type of atom for a given node by name. The overall binding energy is the target property, for the full catalyst process. This means all nodes with identical graph id will have a identical overall binding energy. The x, y, and z coordinates are the coordinates of the nodes in the molecular graph, measured in Ångstrøm.
These fields are then translated to the relevant datastructures, via the graph data set constructor. This constructor defines the following data structures, for utilization in the modelling task. Important to say is that the data set constructor, constructs a data set of popentially several graphs, and not only one. The data set consists of the following critical fields:

- num_nodes: Number of nodes in the data set of graphs
- node_from: The node from which the edge originates
- node_to: The node to which the edge points
- num_graphs: Number of graphs in the data set
- node_graph_index: The index of the graph the edge resides in
- unique_atoms: The number of unique atoms in the data set
- edge_lengths: The lengths of the edges
- edge_vector_diffs: The vector differences of the edges

The number of nodes in dataset, allows for the model to create initial state representation structures with proper dimensions for the modelling task. The node_from property, allows for summation of all neighbouring node state representations, within the cutoff limit, into the designated node, we are trying to model in the message block. The node_to property, allows for initialising the state representation of the node in the update block, namely the scalar properties and the vector properties. The scalar properties are concatenated tensors of embedded atom representations, linked by the index in the node_to property. For scalar properties it is zero-vectors, instead of embedded tensors. The num_graphs property is used to initalize the initial state representation of the graphs. The node_graph_index property is an index linking an edge, to a corresponding graph. When summing over all neighbour nodes occur, the full message state representation of all neighbours under the cutoff, is summed over the node_graph_index property, to produce a state representation of graphs. The unique_atoms property, allows for the embedding function to create a unique atom representation for each type of atom in the data set. The edge_lengths allow for a masking of the edges and thereby nodes that are not within the cutoff limit. The edge_vector_diffs are the vector differences of the edges, being an input variable in the message block.

5.3 Training- and Validation-loop

Algorithm 1 MPNN Training Loop

Initialize size of ensemble: T
Initialize size of batch: B
Initialize data sets: Td, Vd
initialize training data loader: $TD = Dataloader(B, Td)$
initialize validation data loader: $VD = Dataloader(B, Vd)$
Initialize number of epochs: E
Initialize validation index: V
for $model = 1, 2, \ldots, T$ **do**
    Initialize MPNN model instance: $model = PAINN()$
    initialize MSE loss function: $loss = MSE()$
    Initialize Adam optimizer: $optimizer = Adam(model.parameters())$
    Initialize learning rate scheduler: $scheduler = StepLR(optimizer)$
    for $epoch = 1, 2, \ldots, E$ **do**
        for $batch = 1, 2, \ldots, TD$ **do**
            Normalize targets: $y = normalize(y)$
            Predict binding energy: $\hat{y} = model(batch)$
            Calculate loss: $l = loss(y, \hat{y})$
            Backward pass: $l.backward()$
            Optimizer step: $optimizer.step()$
            if $epoch\%V == 0$ **then**
                Model in evaluation mode: $model.eval()$
                for $batch = 1, 2, \ldots, VD$ **do**
                    Normalize targets: $y = normalize(y)$
                    Predict binding energy: $\hat{y} = model(batch)$
                    Calculate loss: $l = loss(y, \hat{y})$
                    Apply exponential smoothing: $l = l * 0.9 + l_{-1} * 0.1$
                    Scheduler step: $scheduler.step(l)$
                end for
            end if
        end for
    end for
end for

5.4 Test Loop

Algorithm 2 MPNN Testing Loop

Initialize size of ensemble: T
Initialize size of batch: B
Initialize data sets: $Testd$,
initialize training data loader: $TestD = Dataloader(B, Testd)$
for $model = 1, 2, \ldots, T$ **do**
    Initialize target list: y_l
    Initialize prediction list: $\hat{y}_l$
    for $batch = 1, 2, \ldots, TestD$ **do**
        Normalize targets: $y = normalize(y)$
        Predict binding energy: $\hat{y} = model(batch)$
        Calculate loss: $l = loss(y, \hat{y})$
        Append target to list: y_l
        Append prediction to list: $\hat{y}_l$
    end for
    Save predictions and targets to file
    Save model to file
end for

5.5 Modelling Hyperparameters

Generally, two subsets of hyperparameters exists, those relevant to the less sophisticated set of ensemble models (titled MPNN64), and those relevant to the more sophisticated ensemble models (titled MPNN128). The table below1, details shared and non-shared model hyperparameters, as well as highlighting the parameters, which are consistent with the PAINN architecture described in [7] with a star (*).

Table 1: Modelling hyperparameters

Model Hyperparameters	***MPNN64***	***MPNN128***	***Unit***
Shared			
Number of physical dimensions*	3		Dimensions
Number of Message Passing Rounds*	5		Rounds
Patience*	5		Validation steps
Weight Decay*	0.01		N/A
Learning Rate Decay*	0.5		N/A
Exponential Smoothing for Validation*	0.9		N/A
Radial Basis Parameter*	20		N/A
Learning Rate	0.004		N/A
Cutoff Distance	3.2		Ångstrøm
Early stopping patience	20		Ångstrøm
Not shared			
State dimension size	64	128	Dimensions

A brief explanation of the individual hyperparameters influence on the model can be found in appendix section10.1. The remaining hyperparameters not chosen based on other experimental settings in related research, were run through an optimization shcheme. The scheme, and its results can be found in the section below.

5.5.1 Hyperparameter selection

Out of all hyperparameters for the MPNN, only three of them were not described in the paper[7]. The three hyperparameters being

- Learning rate
- Cutoff
- Early stopping patience

Out of these three, two of them, learning rate and cutoff, were deemed critical to experiment integrity, and were therefore chosen to go into hyperoptimization trials. The last one, early stopping patience, was chosen based on initial unrecorded experiments in the cloud environment. The exclusion of early stopping patience from the optimization scheme, came down to the difficulty measuring its both positive and negative impact on modelling. Early stopping is implemented both to prevent overfitting to training data, and to create diversity within the ensemble of individual models. Estimating both metrics, was deemed too cumbersome, and therefore excluded, from the optimization scheme. So the learning rate and cutoff were chosen to go into trials. The learning rate, dictates how aggressive the optimization scheme, in the MPNN can be. A higher learning rate meaning more radical changes in model parameters, in response to gradients. Lower meaning less aggresive changes per step. The Cutoff, a locality variable,

dictates the amount of neigbours being included in the message block of the MPNN. The higher the cutoff, the more neighbours will be included in the message block, and thus more information in the model. The lower, the more neighbours will be excluded. A total of two rounds of hyperparameter optimization was run. The first had a broader search space, the latter having a narrower search space. Both sets of trials were run with the *python* library *Hyperopt*[18]. The results of the initial trials can be found in the bottom of the below table2.

Table 2: 1. round hyperparameter optimization

Optimization Scheme	***Value***	***Unit***
Variables		
Learning Rate	[0.0001;0.01]	N/A
Cutoff	[2.0;5.0]	Angstrom
Parameters		
Maximum number of trials	20	trials
Number of graphs trained on	10	graphs
Number of graphs validated on	5	graphs
Epochs per trial	5	epochs
Results		
Learning Rate	**0.0032**	**N/A**
Cutoff	**2.72**	**Angstrom**

Another set of trials were run, with a narrower search space, and more epochs for the models to hopefully converge to a higher degree. The results of the trials can be found in the table below 3.

Table 3: 2. round hyperparameter optimization

Optimization Scheme	***Value***	***Unit***
Variables		
Learning Rate*	[0.0005;0.005]	N/A
Cutoff*	[2.5;3.5]	Angstrom
Parameters		
Maximum number of trials	20	trials
Number of graphs trained on	10	graphs
Number of graphs validated on	5	graphs
Epochs per trial*	10	epochs
Results		
Learning Rate	**0.004**	**N/A**
Cutoff	**3.19**	**Angstrom**

The results indicate a cutoff at 3.19 Angstrom, and a learning rate of 0.004. These hyperparameters were selected for the final run of training both ensembles.

5.6 Experimental Hyperparameters

The experimental hyperparameters defining the implementation of the above model setting, can be found in table below4.

Table 4: Experimental Hyperparameters

Experimental Hyperparameters	***Value***	***Unit***
Number of molecules in Data Set	7.053	Molecules
Training Split	0.8	N/A
Validation Split	0.1	N/A
Test Split	0.1	N/A
Ensemble Size	5	Models
Initial number of Epochs	100	Epochs
Validation Index	5	Epochs
Model Saving Interval	0.01	N/A
Batch Size	20	Molecules

An in explanation of the selection of experiment hyperparameters and justification, be found in the appendix section10.2.

5.7 Software tools and Hardware

The projects execution were developed in Python 3.10.9, with heavy emphasis on Pytorch, Numpy and Pandas, supported by few shell scripts for cloud job definitions. A full list of packages, and their versions utilized in the experiment, can be seen in appendix10.3 The initial datastructures and pipeline were inspired by course-material made by Mikkel Nørgaard Schmidt. The hardware utilized for the final experimental setup, were two two types of GPUs, belonging to the High Performance Computing (HPC) cluster at the Technical University of Denmark (DTU). The two queues utilized were 'gpua100' and 'gpuv100'. For further details on the hardware, please refer to the following reference:[19]. Total computation time in the cloud was three days, in order to produce the trained ensemble of ten individual models.

5.8 Reproduceability

For reviewing and reproduceability purposes, the following github link stores the latest version of the code: `https://github.com/Lohmann94/head`, which was utilized in the experiment. The experiment can be reproduced with the following seed-values5, which can be found for individual models:

Table 5: Model seeds for reproduction

MPNN128 Ensemble:		**MPNN64 Ensemble:**	
Model	***Seed***	***Model***	***Seed***
MPNN128_1	670	**MPNN64_1**	267
MPNN128_2	118	**MPNN64_2**	842
MPNN128_3	600	**MPNN64_3**	91
MPNN128_4	747	**MPNN64_4**	112
MPNN128_5	795	**MPNN64_5**	547

The cloud-job configuration variables for reproducing with the described performance, can be found in the below table6:

Table 6: Cloud Job Parameters

Cloud Job Parameters	***MPNN128***	***MPNN64***
Queue	**gpuv100**	**gpua100**
Number of cores	8	8
Memory/Cores	5 gb	5 gb
Number of hosts	1	1

6 Data

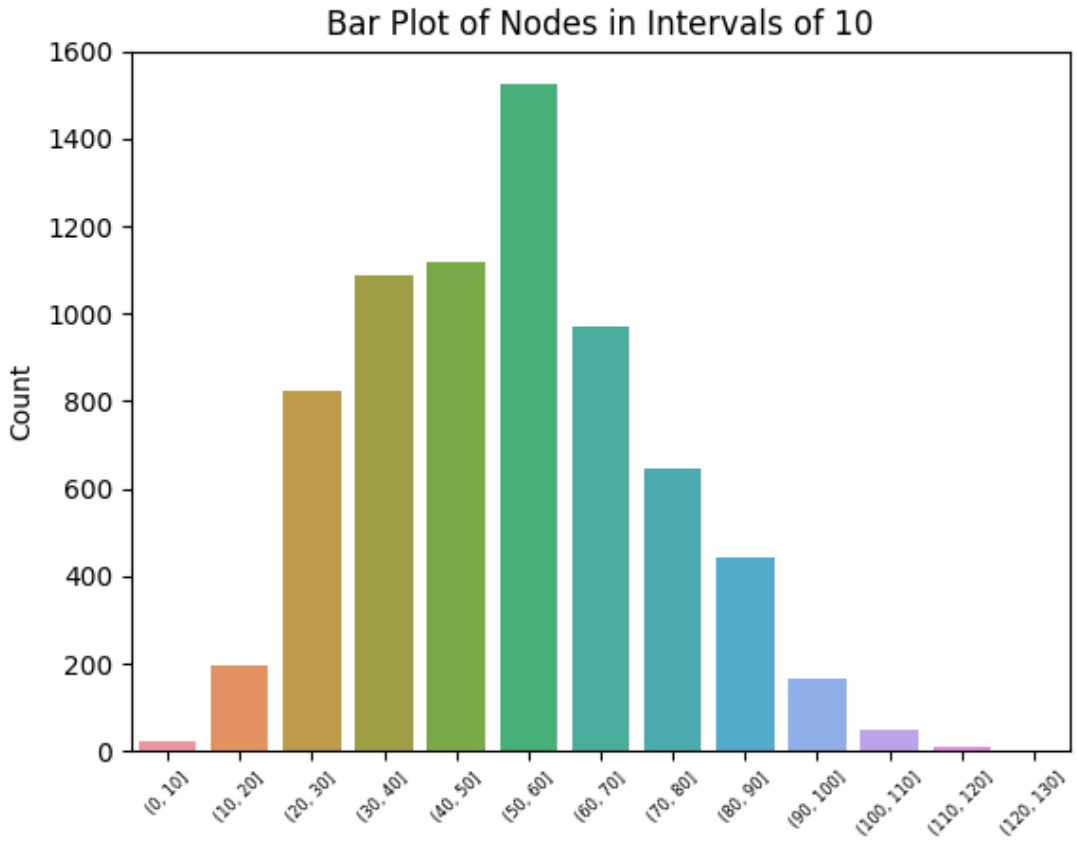
6.1 Cross Coupling Catalysts

The data set of interest consists of 7.054 optimized geometries of compounds being candidate structures in oxidative addition catalystic process. The optimized geometries will have a binding energy associated with it, that dictates the net energy produced by the oxidative addition of a specific transition metal[13], which is the target of our modelling task. The original papers set out to further select 557 catalyst candidates which are within a predefined thermodynamic window, and a further selection of candidates based on their pricepoint per mol.

The molecular compounds are represented as graph structures of nodes and edges. The nodes constitute atoms, and the edges molecular connections between atoms. The representation will further consist of Cartesian coordinates provided in the data set, fixing the individual nodes in three dimensional space. This also provides the data set with the possibility of deriving distances between the nodes, or the length of edges, which are crucial to the modelling efforts.

The number of number of nodes in the graphs, range from a minimum of 5, a mean of 52.51, to a maximum of 123 nodes. A bar plot of the number of graphs in various buckets of node-counts, can be seen below4.

Figure 4: Bar plot of number of nodes per graph



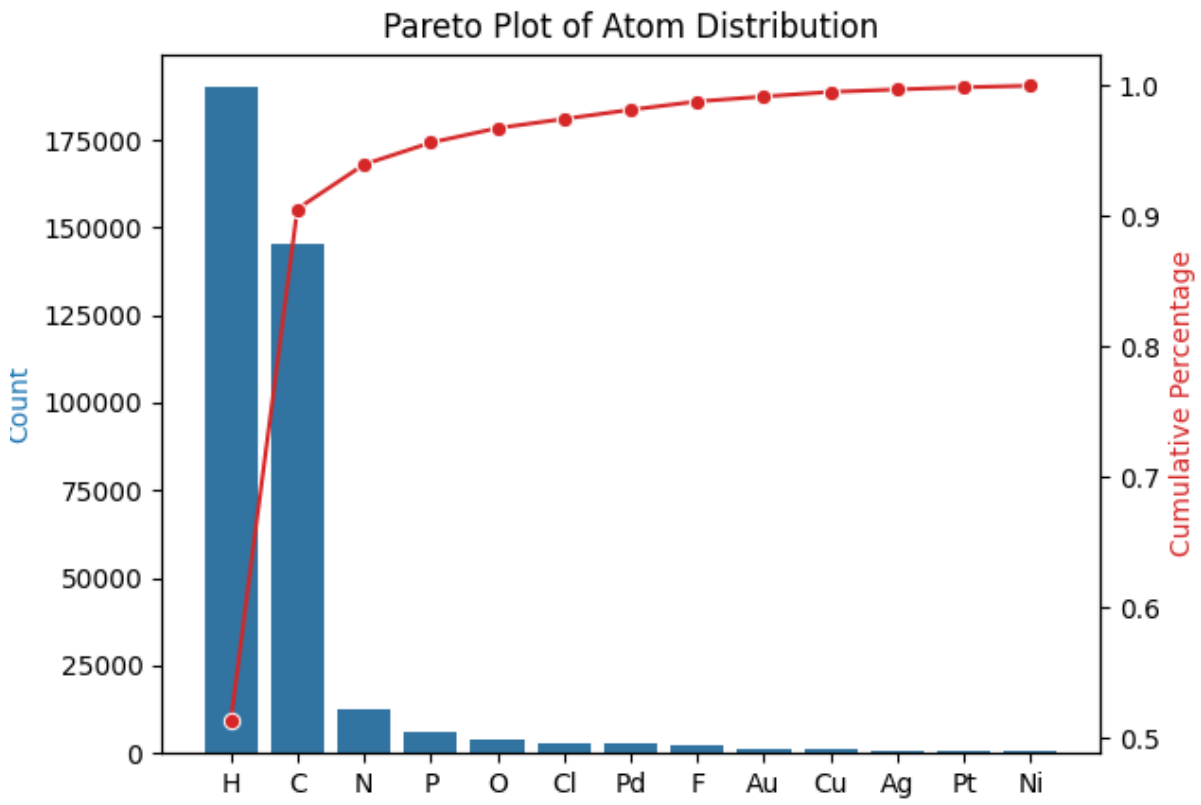
The atom distribution of the graphs a constituted by 13 unique atoms being:

- H: Hydrogen
- C: Carbon
- N: Nitrogen
- P: Phosphorus
- O: Oxygen
- Cl: Chlorine
- Pd: Palladium
- F: Fluorine
- Au: Gold
- Cu: Copper
- Ag: Silver
- Pt: Platinum

- Ni: Nickel

The distribution of the above atoms, can be seen in a Pareto graph below5. The distribution is mainly made up by hydrogen and carbon, those two being 90.6 percent of the atom-representation combined. The next highest contributor is nitrogen, only representing 3,4 percent of the atoms. The remaining atoms constitute 6 percent of the atom representation in the data set.

Figure 5: Pareto plot of atom distribution in data set

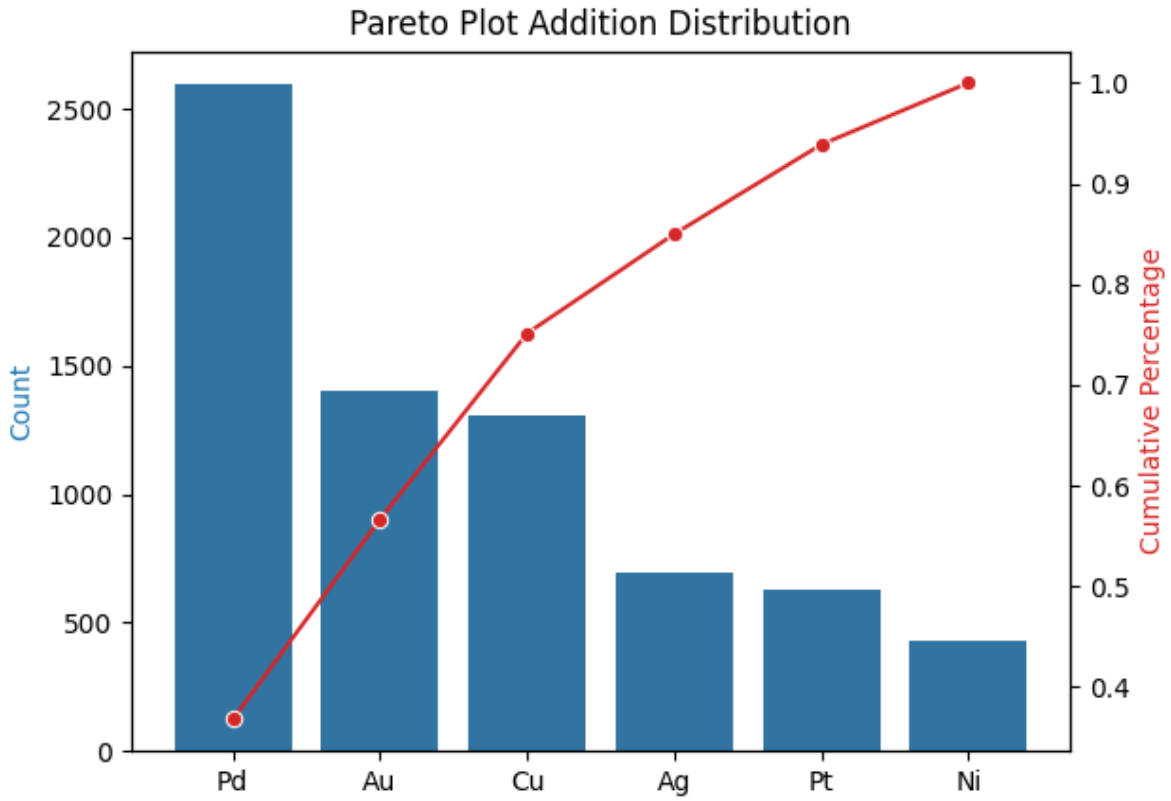


The additive part of the catalyst process, are selected transition metals, namely:

- Pd: Palladium
- Au: Gold
- Cu: Copper
- Ag: Silver
- Pt: Platinum
- Ni: Nickel

The data set consists of the following distribution of the above transition metals, below can be seen a pareto similar to further above, but now show the distribution of transition metals in the data set6.

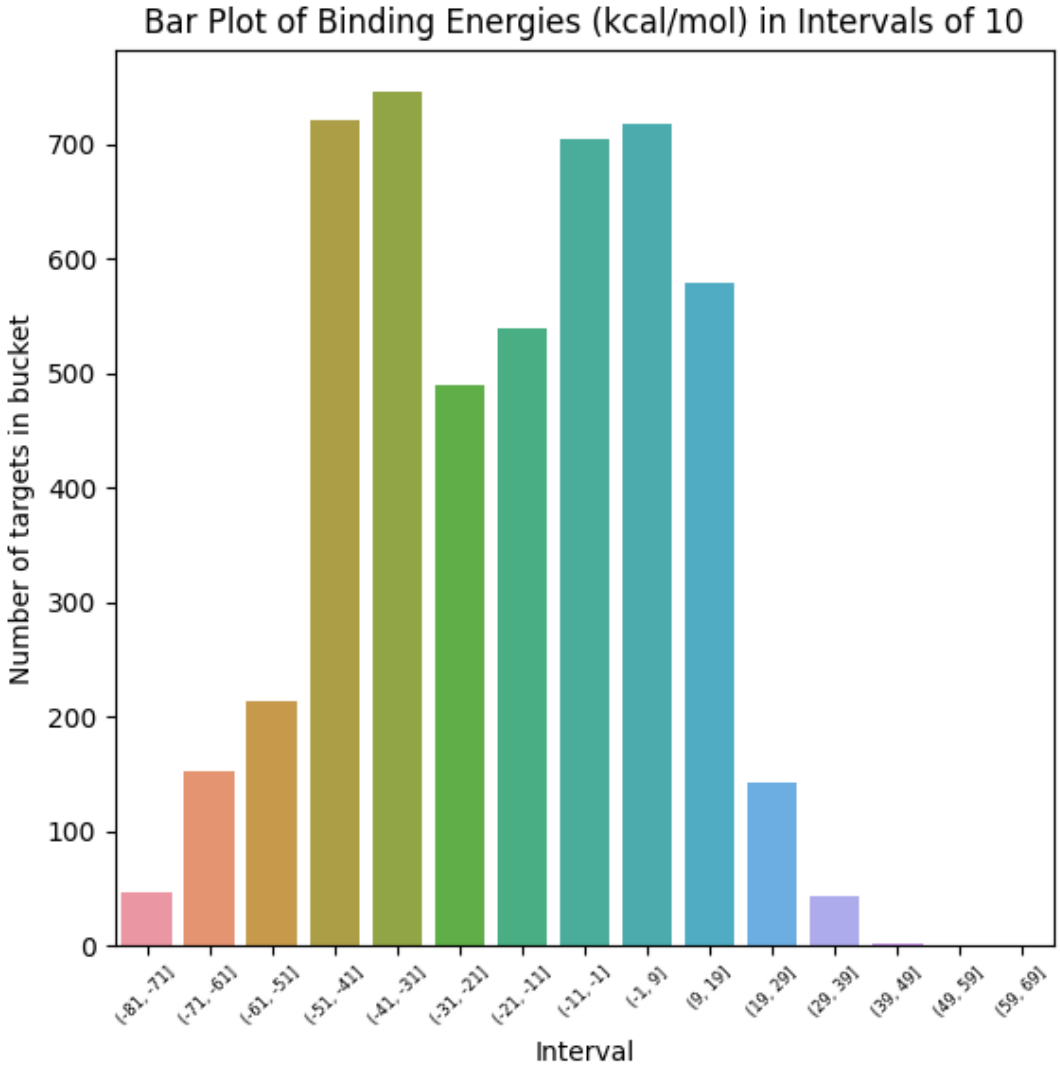
Figure 6: Pareto plot of transition metal distribution in data set



The distribution is mainly comprised of Palladium and Gold, standing for 57 percent of the transition metals in the data set.

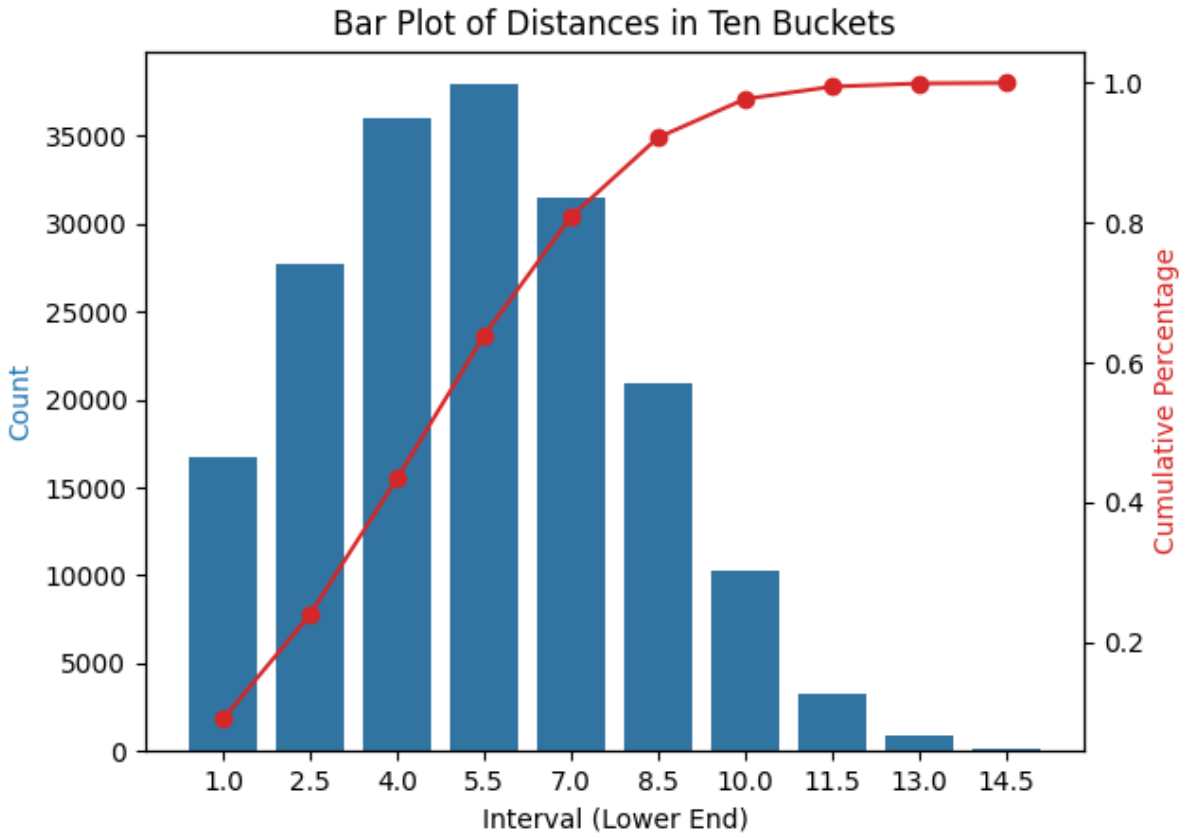
The catalyst process can be both exothermic and endothermic, represented in either a positive net binding energy as a target, or a negative net binding energy as a target for the regression task. The regression targets range from a minimum of -80.82 being endothermic, to 61.49 being exothermic, where the mean is placed at -18.62. A bar plot with buckets of ten, of the distribution of targets can be seen below7.

Figure 7: Bar plot of binding energy distribution in targets



The edge-wise distances between nodes, measured in ångstrom, are crucial to the modelling task, since a model-parameter called 'r-cut', defines a cutoff limit from which edges with a distance lower than the cutoff are defined as neighbours of a given node, and those above as non-neighbours. The neighbours goes are allowed a higher model influence, and therefore understanding where to set the cutoff, is a crucial hyperparameter. The distances of the full data set is intractable to compute locally, so a subset of 50 randomly chosen graphs and their edge-wise distances are plotted as a representation of the full data set8. As we can see in the plot, most distances are covered by the interval 0 to 7, at 64 percent, and only 20 percent of distances lie in the interval above 8.5.

Figure 8: Pareto plot of Distances between edges from 50 graphs



In a chemistry context, it is argued that a large portion in the variance in binding energy, can be explained by local interactions among atoms[7]. So even though a high degree of information lies beyond cutoff limits of i.e four or five, then information might not be crucial to the prediction in energy we are trying to make.

The modelling task will take in a given molecular structure, and produce a regression metric, trying to predict the associated binding energy from the oxidative addition in the unit kcal/mol. Further more, the regression efforts will be put in an ensemble context of a number of structurally similar models, producing a mean and a variance over the predicted regression metric from the ensemble.

6.2 Data Privacy- and Quality-issues

Due to the data sources being publicly available for download, under the creative commons attribution 4.0 license, the implementation and storage of the data, has been chosen to support all project activities in the most convenient way. Specifically that mean local and cloud storage, without password protection or encryption of the raw data.

The field of chemistry, that this study supports computationally, provide research which are used

within the life-science industry for drug-discovery among other fields.

With this in mind, sourcing of the data set needs to be carefully handled, taking into account potential end-user risks of promoting various molecules and processes through research, at some point in the process before consumption. The sourcing of the Cross-Coupling data set, is partly transparent. The initial data set is comprised of 25.116 graph structures, and was sourced through generating options based on 6 transition metals, mentioned above, and 91 ligands. This generation process was made by combining all possible combinations of the six transition metals and the 91 ligands, applying a filter for redundant chemical properties to reach the initial 25.116 size data set[13]. The reduction in data set size to the 7.054, was done by training a model to predict a chemical descriptor value relating to certain energy-levels, and then selecting these 7.054 molecules and transition metals based on that. No apparent precaution towards end-user complications are mentioned, and should therefore be applied further down stream of the research efforts within this field.

7 Results

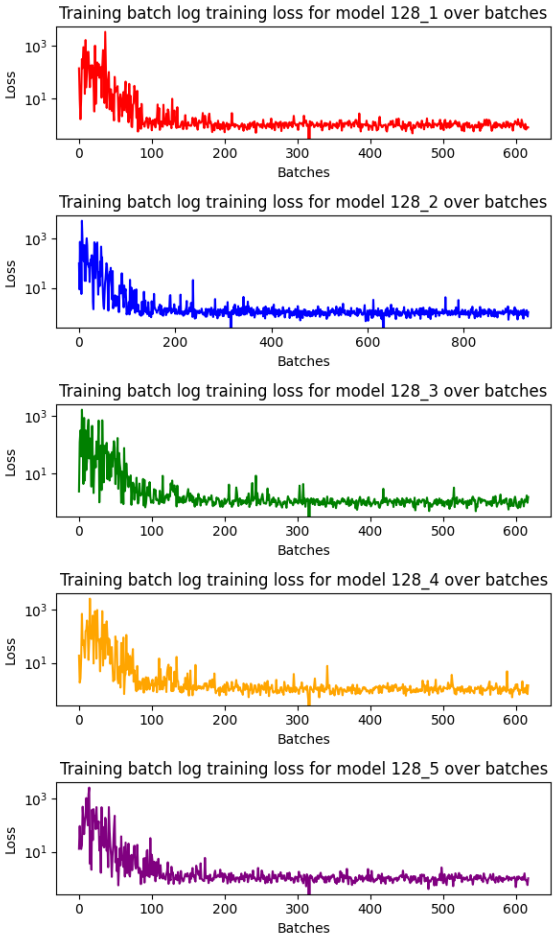
Table 7: Number of gradient steps, for individual models in ensembles

MPNN ensembles total gradient steps			
MPNN128 Ensemble		***MPNN64 Ensemble***	
Model	*Value*	**Model**	*Value*
MPNN128_1	630	**MPNN64_1**	630
MPNN128_2	630	**MPNN64_2**	945
MPNN128_3	630	**MPNN64_3**	630
MPNN128_4	630	**MPNN64_4**	945
MPNN128_5	945	**MPNN64_5**	630

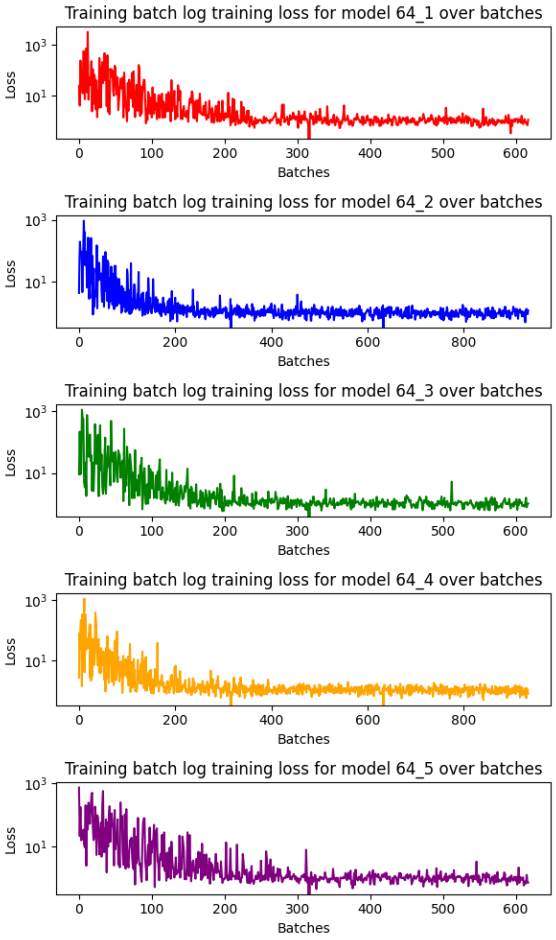
Table 8: Training time for individual models in ensembles, and total training time.

MPNN Total Training Time					
MPNN128 Ensemble			***MPNN64 Ensemble***		
Model	*Value*	*Unit*	**Model**	*Value*	*Unit*
MPNN128_1	383	minutes	**MPNN64_1**	368	minutes
MPNN128_2	473	minutes	**MPNN64_2**	551	minutes
MPNN128_3	384	minutes	**MPNN64_3**	370	minutes
MPNN128_4	455	minutes	**MPNN64_4**	559	minutes
MPNN128_5	668	minutes	**MPNN64_5**	375	minutes
Total minutes	2363	minutes	***Total minutes***	2223	minutes
Total hours	39.38	hours	***Total hours***	37.1	hours

Figure 9: Plots of training loss for individual models in the ensembles of 64-state- and 128-state-representations.

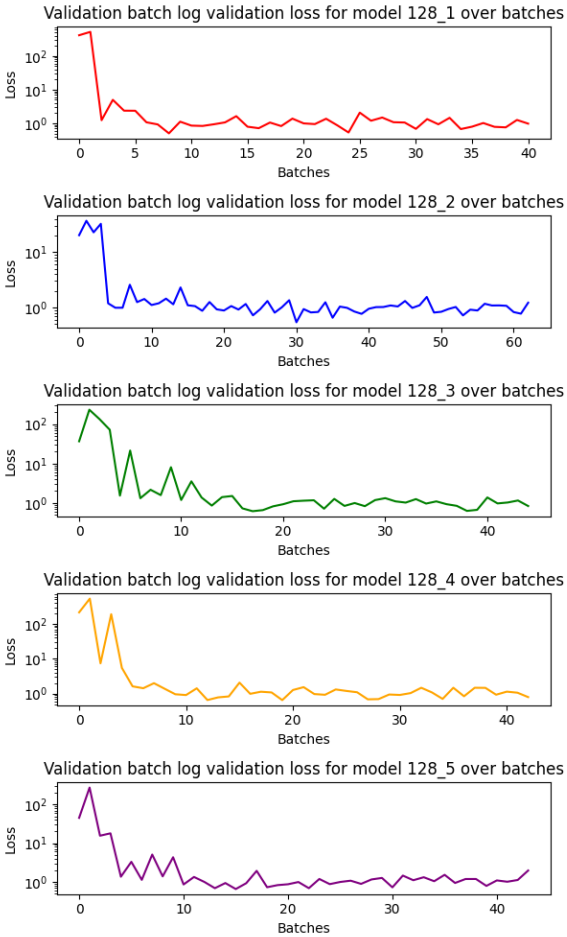


(a) 1a

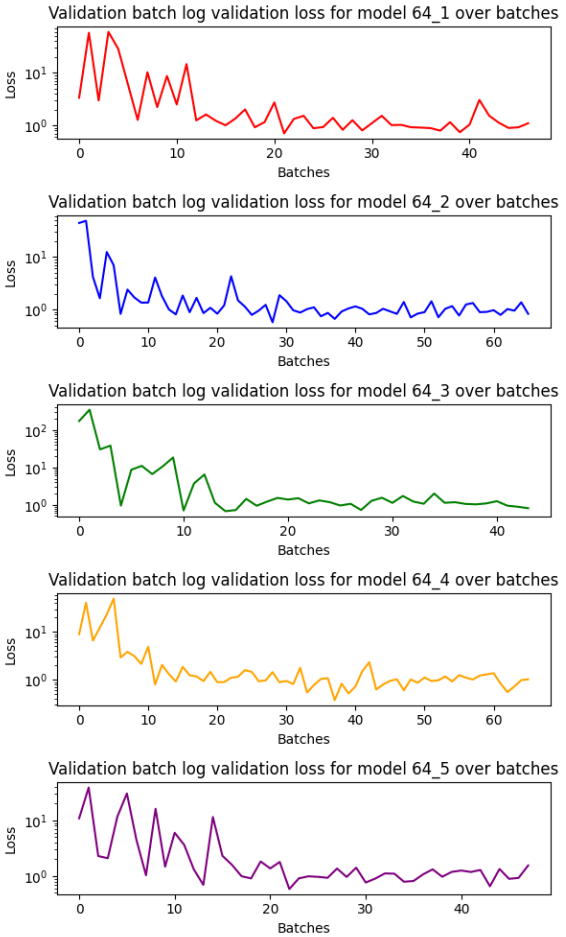


(b) 1b

Figure 10: Plots of training loss for individual models in the ensembles of 64-state- and 128-state- representations.



(a) 1a



(b) 1b

8 Discussion

9 Conclusion

References

[1] J. Busk, P. B. Jørgensen, A. Bhowmik, M. N. Schmidt, O. Winther, and T. Vegge, "Calibrated uncertainty for molecular property prediction using ensembles of message passing neural networks," *Machine Learning: Science and Technology*, vol. 3, 7 2021. [Online]. Available: https://arxiv.org/abs/2107.06068v2

[2] J. Behler, "Atom-centered symmetry functions for constructing high-dimensional neural network potentials," *Journal of Chemical Physics*, vol. 134, p. 74106, 2 2011. [Online]. Available: /aip/jcp/article/134/7/074106/954787/Atom-centered-symmetry-functions-for-constructing

[3] J. Gasteiger, J. Groß, and S. Günnemann, "Directional message passing for molecular graphs," *8th International Conference on Learning Representations, ICLR 2020*, 3 2020. [Online]. Available: https://arxiv.org/abs/2003.03123v2

[4] K. Atz, F. Grisoni, and G. Schneider, "Geometric deep learning on molecular representations," *Nature Machine Intelligence*, vol. 3, pp. 1023–1032, 12 2021.

[5] O. T. Unke, M. Bogojeski, M. Gastegger, M. Geiger, T. Smidt, and K. R. Müller, "Se(3)-equivariant prediction of molecular wavefunctions and electronic densities," *Advances in Neural Information Processing Systems*, vol. 18, pp. 14 434–14 447, 6 2021. [Online]. Available: https://arxiv.org/abs/2106.02347v2

[6] S. Zhang, Y. Liu, and L. Xie, "Molecular mechanics-driven graph neural network with multiplex graph for molecular structures," 11 2020. [Online]. Available: https://arxiv.org/abs/2011.07457v1

[7] K. T. Schütt, O. T. Unke, and M. Gastegger, "Equivariant message passing for the prediction of tensorial properties and molecular spectra," *Proceedings of Machine Learning Research*, vol. 139, pp. 9377–9388, 2 2021. [Online]. Available: https://arxiv.org/abs/2102.03150v4

[8] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 12 2018. [Online]. Available: https://arxiv.org/abs/1812.08434v6

[9] H. Song, T. Diethe, M. Kull, and P. Flach, "Distribution calibration for regression," *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 10 347–10 356, 5 2019. [Online]. Available: https://arxiv.org/abs/1905.06023v1

[10] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods," *Machine Learning*, vol. 110, pp. 457–506, 3 2021.

[11] G. Scalia, C. A. Grambow, B. Pernici, Y. P. Li, and W. H. Green, “Evaluating scalable uncertainty estimation methods for dnn-based molecular property prediction,” *Journal of Chemical Information and Modeling*, vol. 60, pp. 2697–2717, 10 2019. [Online]. Available: https://arxiv.org/abs/1910.03127v1

[12] T. Pearce, F. Leibfried, A. Brintrup, M. Zaki, and A. Neely, “Uncertainty in neural networks: Approximately bayesian ensembling,” *Proceedings of Machine Learning Research*, vol. 108, pp. 234–244, 10 2018. [Online]. Available: https://arxiv.org/abs/1810.05546v5

[13] B. Meyer, B. Sawatlon, S. Heinen, O. A. V. Lilienfeld, and C. Corminboeuf, “Machine learning meets volcano plots: Computational discovery of cross-coupling catalysts,” *Chemical Science*, vol. 9, pp. 7069–7077, 2018.

[14] B. Lakshminarayanan, A. Pritzel, and C. Blundell, “Simple and scalable predictive uncertainty estimation using deep ensembles,” *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 6403–6414, 12 2016. [Online]. Available: https://arxiv.org/abs/1612.01474v3

[15] K. Tran, W. Neiswanger, J. Yoon, Q. Zhang, E. Xing, and Z. W. Ulissi, “Methods for comparing uncertainty quantifications for material property predictions,” *Machine Learning: Science and Technology*, vol. 1, 12 2019. [Online]. Available: https://arxiv.org/abs/1912.10066v2

[16] “The radial basis function kernel.”

[17] K. Schütt, P.-J. Kindermans, H. E. S. Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller, “Schnet: A continuous-filter convolutional neural network for modeling quantum interactions,” *Advances in Neural Information Processing Systems*, vol. 30, 2017. [Online]. Available: www.quantum-machine.org.

[18] “Hyperopt documentation.” [Online]. Available: https://hyperopt.github.io/hyperopt/

[19] “Using gpus under lsf10.” [Online]. Available: https://www.hpc.dtu.dk/?page_id=2759

10 Appendix

10.1 Appendix A

10.2 Appendix B

10.3 Appendix C

Two lists of main packages, and their versions vital to local development and cloud execution. First the local development packages10.3:

- Python 3.10.9
- Pytorch 2.0.1+cu118
- numpy 1.23.5
- pandas 1.5.2
- matplotlib 3.7.0
- tqdm 4.65.0
- python-dotenv 1.0.0

And for cloud execution10.3:

- Python 3.9.14
- Pytorch 2.1.0+cu118
- numpy 1.23.5
- pandas 1.5.2
- scipy 1.9.1bstat
- tqdm 4.66.1