UNIVERSITY OF ZAGREB
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
DEPARTMENT OF PHYSICS

Fran Krčelić

Multi-fidelity machine learning of material properties

Mentor: dr. sc. Ivor Lončarić

Master's thesis

Zagreb, 2024.

First of all, I would like to thank my mentor, Dr. Ivor Lončarić, for his patience and enthusiasm which he shared with me. I am also grateful for his advice and the trust he placed in me.
Above all, I would like to thank my family and friends, who constantly motivated me and provided support throughout my entire studies.

Abstract

This thesis explores the application of equivariant graph neural networks with random Fourier features in predicting molecular energy. By combining two datasets of different accuracies, ANI-1x and ANI-1ccx, consisting of energies of organic molecules based on H, C, N, and O elements, the model aims to enhance predictive power through a deeper understanding of interatomic interactions. ANI-1x contains data generated with density functional theory for molecules selected through active learning, which allows for better generalization with fewer data points, while ANI-1ccx provides highly accurate energy values calculated using coupled-cluster methods. Training with various ratios of these datasets demonstrated that increasing the amount of ANI-1x data in the training set results in improved predictive power of the model. Additionally, the application of this approach to more complex systems, such as molecular crystals, was discussed. This method opens new possibilities for more accurate modeling in physics, chemistry and materials science.

Keywords: neural networks, equivariant graph neural networks, random Fourier features, molecular energy

Contents

1 Introduction

Materials physics is an interdisciplinary field that spans a wide range of spatial scales. The most fundamental of these is the observation of individual atoms and the three-dimensional structures they form by creating chemical bonds. A precise description and understanding, and ultimately manipulation of materials at the atomic level, is the central problem of materials physics.

Typically, we are interested in the energies and forces acting on individual atoms in order to model the system. For this purpose, a useful physical quantity is the potential energy surface (PES). It depends solely on the positions of atoms and is determined using quantum mechanics. The problem therefore reduces to solving the Schrödinger equation for the electrons in question (the use of PES implies the Born–Oppenheimer approximation). Unfortunately, an exact solution of such a problem cannot be obtained even with the most sophisticated computers available today. Consequently, computational simulations at the atomic level have long been used to approximately solve the problem, with two main approaches standing out.

The first of these methods is based on density functional theory (DFT) [1]. Although this method yields excellent results, it is limited by the scale of the system. Its complexity grows with the third power of the number of electrons in the system, making it applicable up to scales of a few hundred atoms. Nevertheless, using DFT, new compounds and previously unknown structures have been identified, which were later experimentally realized.

On the other hand, the idea is to parametrize interatomic interactions using a generalized potential, the so-called force field. The potential is described by well-known simple terms (harmonic contribution, Coulomb interaction, etc.), whose parameters are then fitted to the system, yielding an empirical model. Expectedly, this procedure sacrifices part of the accuracy, but in return it provides speed, with model complexity scaling quadratically with the number of atoms. Such simulations extend the scale of systems to several million atoms [2]. Although useful for some systems, such as describing DNA molecules, force fields are not sufficiently complex potentials to achieve the accuracy required for a broader class of materials.

As in many other fields in recent years, machine learning has found applications here as well. In general, there are three main types of machine learning:

- **Supervised learning** is a method of training a model on a predefined dataset where the outcome is known. This type of learning is used as a form of regression. The task of the model is to make predictions based on the given data, which are evaluated using a loss function. Training ends when the deviations of predictions from actual results become satisfactorily small.

- **Unsupervised learning** refers to cases where the model trains on a dataset but the outcome is not *a priori* known. The goal is to uncover hidden relationships and patterns in the dataset without the need for a specified target.

- **Reinforcement learning** is a method in which the model interacts with an environment and receives feedback in the form of positive or negative reinforcement. This type of learning is used in robotics and the video game industry. The goal is to teach the network to perform actions that maximize the final reward.

In this work, we employ supervised learning. The idea is to determine the PES using neural networks (NNs). The first such attempt for large systems was made by Smith in 1999 [3]. The idea was generalized by Behler and Parrinello in 2007 in the case of silicon [4]. Since then, applications have been extended to many other systems [5–7]. Moreover, it has been shown that deep NNs are capable of achieving the accuracy of *ab initio* methods, while being up to five orders of magnitude faster [8]. However, the limiting factor is that the complexity of the configuration space grows exponentially with the number of different chemical species [9].

The process is similar to that of force fields, since certain assumptions are also made, such as locality and smoothness of the potential function. The key difference, however, is that there are no assumptions about the form of the interaction, i.e., the dependence of the potential on some variable (e.g., atomic position) in a predefined way. Once the potential is fitted to the data, we can predict energies and forces on large atomic scales without requiring new information. Furthermore, by knowing the PES, we gain access to all quantities that depend on it.

The drawback of this approach arises from the same reason as its advantage. By fitting the potential to specific data, we face the problem of not knowing the sufficient amount of data required for the network to provide satisfactory results. In addition, the physicality of the system is not imposed *a priori*, but instead the system "learns" the physics of the problem on its own, which can lead to nonsensical results.

Due to the aforementioned complexity of networks with respect to the number of different atoms, for the purposes of this work we use only four elements (H, C, N, O). By combining these four elements, we can construct a large portion of organic compounds, which are the focus here. Our goal is to investigate the method of combining data of varying accuracy in order to obtain a better model compared to using only a smaller amount of “higher-quality” data.

2 Methodes

The first ideas of neural networks began to emerge in the second half of the last century. They are a type of machine learning inspired by the functioning of the brain. The basic unit of a network is the *neuron*, which receives information, processes it, and then passes it on to the next neuron. The contribution of an individual neuron is determined by its *weights*, which are adjustable numerical parameters. By the term "training the network," we mean the modification of weights in such a way that the best final model is obtained. This condition is imposed by minimizing the mean squared error (MSE), which measures the deviation of the model from already known values. Since the MSE is directly related to the weights, by adjusting them we adapt the network, and in this case we say that it "learns" the properties of the material.

Neurons are grouped into layers, each consisting of a set of neurons that do not communicate with each other but interact with neighboring layers. In this way, the output of one layer becomes the input to the next. Layers are divided into three groups: input, hidden, and output layers (see Fig. 2.5). The input layer receives raw data, while the output layer produces the final result. In contrast, there may be more than one hidden layer, which is usually the case. Hidden layers are responsible for the computational capability of the network. If the network contains more than one hidden layer, it is referred to as a deep neural network (DNN).

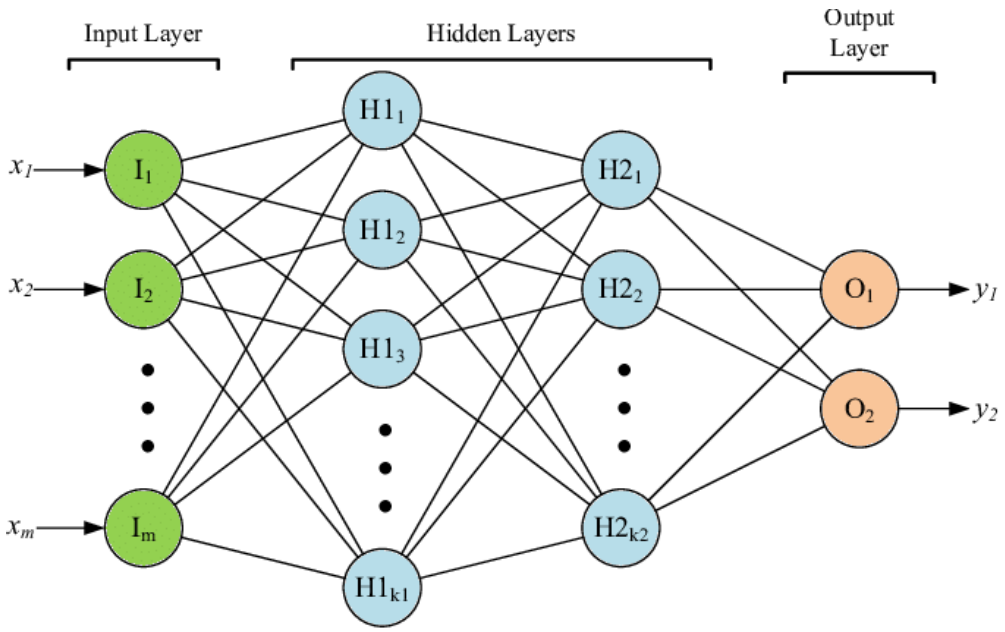

Figure 2.1: Schematic representation of a neural network.

Let us consider what actually happens during information processing within each neuron. As mentioned, the output of the previous layer is in fact the input of the next one, i.e., a neuron receives an N-dimensional vector $\mathbf{x}$, where N is the number of neurons in the previous layer. The first step is its linear transformation:

$$z = \mathbf{w} \cdot \mathbf{x} + b, \tag{2.1}$$

where $\mathbf{w}$ is the previously mentioned N-dimensional weight vector of the neuron, and b is the bias term. If this were the only step, the final result would simply be a linear transformation of the input vector. Since the composition of two linear transformations is also a linear transformation, no matter how many layers we use, the final result could always be represented as one linear operation on the input parameters. In that case, the network would be limited, as it could not describe more complex dependencies. For this reason, a second step in processing is required.

Nonlinearity is introduced by evaluating a nonlinear function at the point z. This procedure enables the network to describe more complex dependencies than purely linear ones. There are many nonlinear functions used for this purpose; however, which one to use and when is beyond the scope of this work. More information on this topic can be found in other sources [10]. Some of the most commonly used nonlinear functions are the rectified linear unit (ReLU, Fig. 2.2 left) and the sigmoid function (Fig. 2.2 right).

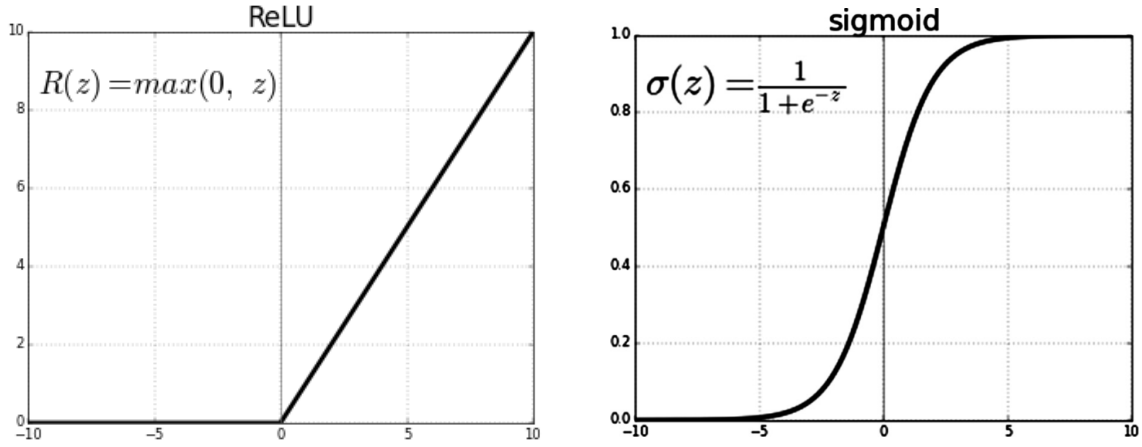

Figure 2.2: Activation functions: ReLU (left) and sigmoid (right).

Now that we understand the process occurring in an individual neuron, let us examine how the weights are adjusted, i.e., how the network is "trained." Since in this work we use supervised learning, the role of regression is played by the previously

mentioned MSE:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2. \tag{2.2}$$

Here, Y_i represents the true value of the variable we are trying to predict, $\hat{Y}_i$ the predicted value from our model, and n the number of examples used for prediction.

It remains to be clarified which values constitute the input layer of the network. Since we are dealing with supervised learning, we provide the network with ordered pairs (X, Y), where X is the input vector and Y is the vector of true, pre-computed values. More specifically, in this case X would contain the atomic positions and their chemical symbols, while Y would represent the energy of that molecule computed by another method, e.g., density functional theory (DFT). A collection of such pairs is referred to as the training set.

One might naively think that the format in which we present the data to the network is practically irrelevant, as long as all the information is encoded. For instance, should we use Cartesian or spherical coordinates, or some other representation? It turns out that neither of these two options is suitable. What we expect is invariance of the system to rotations and translations, as well as to the exchange of atoms of the same element. The solution to this problem lies in the use of graphs to represent molecules. In this way, explicit atomic coordinates are eliminated in favor of emphasizing the connections between them.

2.1 Behler-Parrinello neural network

Among the first to solve the problem of system representation were Behler and Parrinello in their 2007 paper [4]. Their idea was to transform Cartesian coordinates into symmetry functions (SFs) G_i^{μ}, which then satisfy the required properties. The index i is associated with the index of the atom, while μ takes the value 1 for the radial part and 2 for the angular part of the function. The final expression for the radial part is

$$G_i^1 = \sum_{j\neq i}^{\text{atoms}} e^{-\eta\left(R_{ij}-R_s\right)^2} f_c\left(R_{ij}\right). \tag{2.3}$$

The parameter η represents the width of the Gaussian, while R_s is the value around which it is centered. The function f_c is the cutoff function, whose role is to ensure that only atoms within a certain neighborhood of the considered atom contribute to the sum. Its explicit form is

$$f_c\big(R_{ij}\big) = \begin{cases} \dfrac{1}{2}\cos\left(\dfrac{\pi R_{ij}}{R_C}\right) + \dfrac{1}{2}, & \text{for } R_{ij} < R_C \\ 0, & \text{for } R_{ij} > R_C. \end{cases} \tag{2.4}$$

This form ensures the continuity of the function as well as the continuity of its first derivative. The parameter R_C is called the cutoff radius; in other words, only atoms within a sphere of radius R_C contribute to the sum of Gaussians. Such a transformation of coordinates guarantees that the functions satisfy the required symmetry properties, since they depend only on relative atomic distances and contain all the essential information about interactions between neighboring atoms.

The angular part of the symmetry functions has the form

$$G_i^2 = 2^{1-\zeta} \sum_{j,k\neq i}^{\text{atoms}} \big(1 + \lambda\cos\theta_{ijk}\big)^{\zeta} e^{-\eta\left(R_{ij}^2 + R_{ik}^2 + R_{jk}^2\right)} f_c\big(R_{ij}\big) f_c\big(R_{ik}\big). \tag{2.5}$$

Here, the summation involves atomic triplets i, j, and k, with parameters ζ, λ, and η that are not unique and can be tuned as needed. The angle within the cosine is defined as

$$\theta_{ijk} = \frac{\overrightarrow{R}_{ij} \cdot \overrightarrow{R}_{ik}}{R_{ij} R_{ik}}. \tag{2.6}$$

Just like the radial function, this function depends only on relative atomic distances (R_{ij}) and therefore fulfills the same symmetry requirements. Of course, this is not the only possible choice of functions satisfying invariance properties; many other functions with the same properties exist.

Having established a system representation through symmetry functions, the next step is to calculate the energies of individual atoms.

The functions G_i^μ become the input parameters of neural networks S_i, while the output parameters represent the desired atomic energies E_i. By summing over all atomic energies, we obtain the total energy of the system:

$$E = \sum_i E_i. \tag{2.7}$$

A schematic of the network is shown in Fig. 2.3.

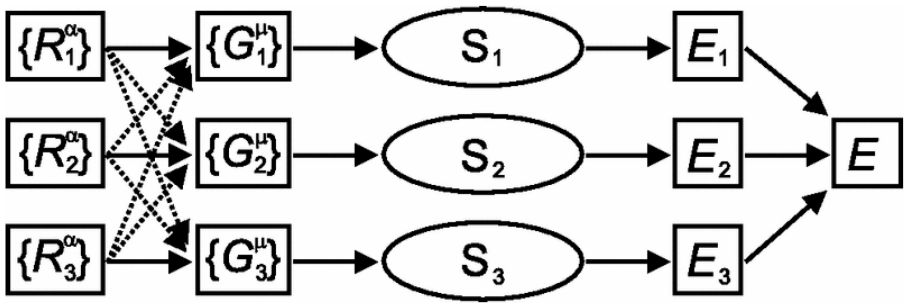

Figure 2.3: Schematic representation of the Behler–Parrinello neural network, adapted from [4].

It is important to note that if two or more atoms in the system belong to the same chemical element, the initial weights of the networks S_i are set to the same values in order to preserve invariance under permutations of those atoms.

2.2 ANI-1

Although the Behler–Parrinello neural network had comparable speed and even better predictive power than existing force fields, there was still much room for improvement. One of the major issues was the adaptability of the model. Their symmetry functions (SFs) were applicable only to a single chemical system. The authors of the new ANAKIN-ME (Accurate NeurAl networK engINe for Molecular Energies) model [8], or simply the ANI model, identified two main limitations of the BP model.

First, SFs lack a functional form capable of capturing recognizable features (spatial arrangements of atoms typically found in common organic molecules, e.g., benzene rings, alkenes, functional groups) in the molecular representation. This shortcoming prevents the neural network from learning interactions within one molecule and transferring that knowledge to another. Second, SFs have limited differentiation of atomic numbers, which empirically makes training difficult in complex chemical

environments. Overall, the combination of these issues restricts the original SFs to chemically symmetric systems with one or two types of atoms or to very small single-molecule datasets.

The authors of the ANI model replaced SFs with so-called atomic environment vectors (AEVs), which retained the main ideas of the BP model but introduced several modifications that improved the approach. A schematic of the network is shown in Fig. 2.4.

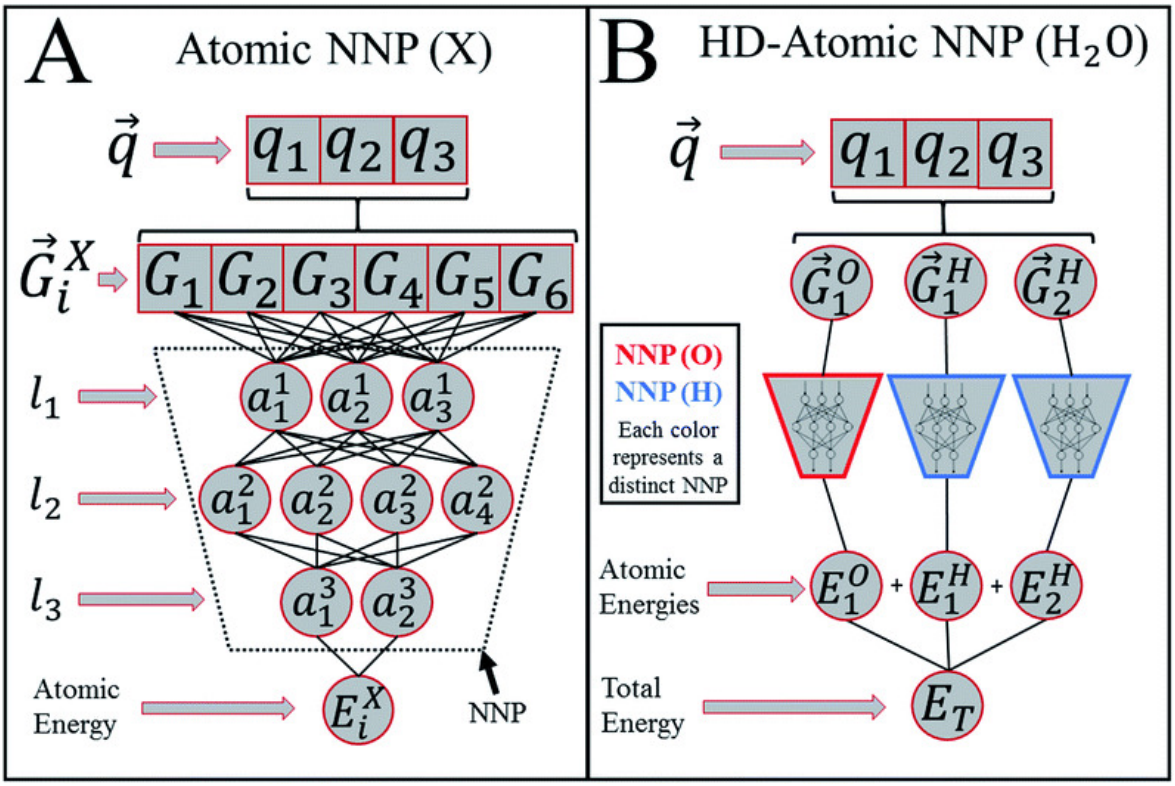

Figure 2.4: Schematic representation of the ANI-1 neural network, adapted from [8].

The radial component retained the same form as before:

$$G_m^R = \sum_{j \neq i}^{\text{atoms}} e^{-\eta\left(R_{ij}-R_s\right)^2} f_c\left(R_{ij}\right), \tag{2.8}$$

where f_c is defined as in Eq. 2.4. The difference lies in the index m, which is in fact an ordered pair (η, R_s). The role of the parameters is the same as before: η defines the width of the Gaussian and R_s the value around which it is centered. The novelty is that these parameters are no longer fixed. More precisely, R_s is variable and changes from atom to atom, while the Gaussian width is kept fixed for network stability, though it can still be modified if necessary.

More significant changes were introduced in the angular part of the AEVs:

$$G_m^{A_{\text{mod}}} = 2^{1-\zeta} \sum_{j,k\neq i}^{\text{atoms}} \big(1 + \cos(\theta_{ijk} - \theta_s)\big)^{\zeta} \exp\left[-\eta\left(\frac{R_{ij} + R_{ik}}{2} - R_s\right)^2\right] f_c\big(R_{ij}\big) f_c\big(R_{ik}\big). \tag{2.9}$$

Here, the index m represents an ordered quadruple of parameters $(\zeta, \theta_s, \eta, R_s)$. The newly introduced parameter θ_s allows for an arbitrary phase shift, while R_s has a role similar to that in the radial part. Although these modifications may seem minor, they greatly increase the flexibility of the network, enabling it to predict energies for molecules larger than those in the training set. The dependence of symmetry functions on the chosen parameters is shown in Fig. 2.5.

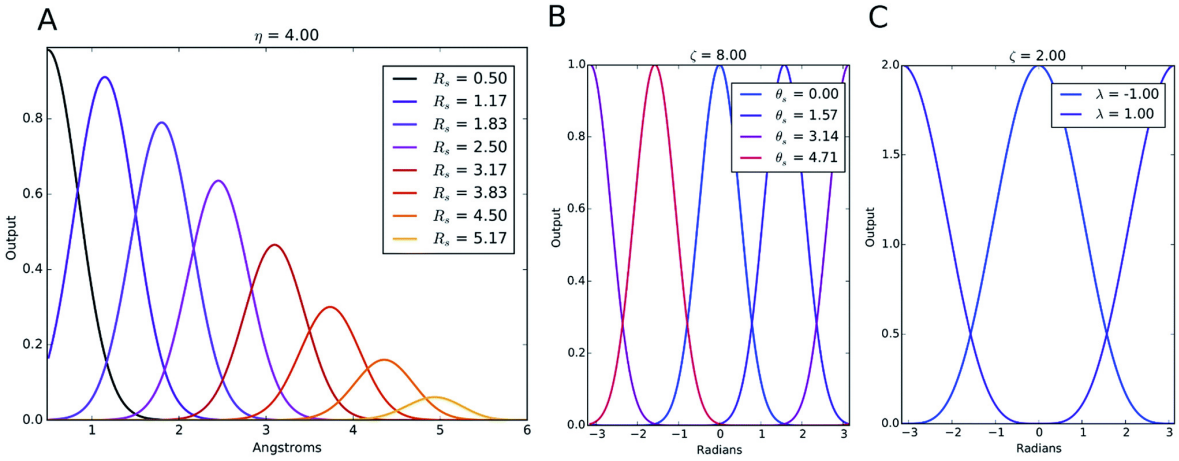

Figure 2.5: Dependence of SFs on different parameters, adapted from [8]. (A) Radial SFs, (B) Modified angular SFs, (C) Original Behler–Parrinello angular SFs.

The next problem was to select a suitable dataset for training the network. The authors chose the GDB-11 database [11, 12], which contains all molecules with up to 11 atoms of the elements C, N, O, and F. For the purposes of ANI, molecules containing fluorine were excluded, and the number of "heavy" atoms was limited to 8. This left 57,951 molecules suitable for training the network. This set was not large enough, but the authors devised a clever solution. Since all molecules were originally given in their ground state, the idea was to generate new examples by exciting normal modes of vibration. Hence the name normal mode sampling (NMS).

The first step of this procedure is to calculate the normal coordinates of a molecule. If a molecule has N_a atoms, then the normal coordinates are

$$Q = \{q_1,\, q_2,\, q_3,\, ...,\, q_{N_f}\}. \tag{2.10}$$

The parameter N_f denotes the number of vibrational modes, which is $N_f = 3N_a - 5$ for linear molecules and $N_f = 3N_a - 6$ for all others. The normal coordinates were computed using one of the desired *ab initio* methods. Alongside them, the corresponding force constants were calculated:

$$K = \{K_1,\, K_2,\, K_3,\, ...,\, K_{N_f}\}. \tag{2.11}$$

It remained to generate displacements of atoms from the equilibrium state. For this purpose, a set of pseudo-random numbers c_i was generated such that

$$\sum_{i}^{N_f} c_i \in [0, 1], \tag{2.12}$$

and these were then used to generate displacements for each normal coordinate:

$$R_i = \pm\sqrt{\frac{3c_i N_a k_b T}{K_i}}, \tag{2.13}$$

such that the harmonic potential corresponds to the average energy of a particle system at a given temperature T, scaled by c_i. The sign of the displacement was determined using a Bernoulli distribution with $p = 0.5$, ensuring that both sides of the harmonic potential were equally represented. The normal coordinates were then scaled with the obtained displacement, $q_i^R = R_i q_i$, and new conformations were generated by shifting the normal coordinates by the corresponding value:

$$Q_i^R = \{q_1 + q_1^R,\, q_2 + q_2^R,\, ...,\, q_{N_f} + q_{N_f}^R\}. \tag{2.14}$$

Finally, the energies of all molecular conformations were calculated using the ωB97x exchange–correlation functional [13] with the 6-31G(d) basis set [14]. Although more accurate methods exist, their computational cost is too high for this volume of data. In this way, the original dataset was expanded nearly 300-fold, yielding approximately 17.2 million conformations.

It is also important to mention the network architecture and the loss function used. Empirically, the best-performing architecture was $768 : 128 : 128 : 64 : 1$,

i.e., 768 input parameters, three hidden layers, and one value in the output layer representing the system energy. The best ANI potential obtained with this network was named ANI-1. During training, an exponential loss function was used:

$$C\big(\overrightarrow{E}^{\text{ANI}}\big) = \tau \exp\left(\frac{1}{\tau}\sum_{j}\left(E_j^{\text{ANI}} - E_j^{\text{DFT}}\right)^2\right), \tag{2.15}$$

where $\overrightarrow{E}^{\text{ANI}}$ is the vector of predicted energies E_j^{ANI}, and E_j^{DFT} are the corresponding reference energies calculated with DFT.

Although trained on a dataset containing only up to 8 "heavy" atoms, ANI-1 proved applicable to systems twice as large. This potential achieved nearly the same accuracy as the DFT method used for training, while being up to six orders of magnitude faster in computing atomic energies and forces. Furthermore, it was empirically shown that the numerical complexity per atom for large molecules scales equivalently to that of force fields.

2.3 ANI-1x

After the ANI-1 model, the question arose of how to further improve the predictive power of this neural network. An obvious proposal would be to increase the complexity of the network; however, this is closely tied to the required amount of training data, which results in longer training times. The idea that proved successful was, in fact, to reduce the training set, but in a smart way. By employing so-called active learning (AL), the authors of the ANI-1 model achieved better performance using only 25% of the original dataset, and the new model was named ANI-1x [15].

The existing training dataset contained many conformations that did not contribute any new information to the network and only slowed down the training process. The idea of AL is to select, *a priori*, the examples that will constitute the training set. This procedure consists of two main stages.

The first step was to separate 2% of the data from the original ANI-1 set and train the model on it. The remaining molecules were then tested, and 2% of discarded examples were added to the new training set. A molecule was considered discarded if the model made large errors in predicting its energy, with the exact criterion given by the condition

$$\left|E_{\text{ANI}} - E_{\text{DFT}}\right|/\sqrt{N} > 0.04\,\text{kcal/mol}, \tag{2.16}$$

where N is the number of atoms in the considered molecule. This process was repeated iteratively as long as more than 5% of the examples in the test set satisfied condition (2.16). Once this was no longer the case, the remaining $< 5\%$ of examples were manually added to the training set. All parameters used were arbitrary, and the process could be performed in finer cycles at the cost of additional training time.

In the second stage, the configurational space was explored through random sampling of examples unseen by the model. For this purpose, the authors used the GDB-11 [11, 12] and ChEMBL [16–18] datasets. An ensemble of five ANI-1 models was employed, and the energy for each was computed. If the models agreed, it was an indication that the system was well described and thus not of high importance. Conversely, if the models gave inconsistent results, adding such a molecule to the training set was beneficial. This procedure is known as query by committee (QBC) [19].

Agreement among models in the ensemble was quantified using the standard deviation of the predictions, normalized by the square root of the number of atoms in the molecule:

$$\rho_i = \frac{\sigma_i}{\sqrt{N_i}}. \tag{2.17}$$

If this value exceeded a predefined threshold $\hat{\rho}$, the molecule was included in the training set. To determine $\hat{\rho}$, the following quantity was introduced:

$$\varepsilon_i = \left| MAX\left(\left\{ E_T^{\text{ANI}} \right\}_i^{\text{ens}} - E_{T,i}^{\text{REF}} \right) \right| / \sqrt{N_i}, \tag{2.18}$$

which represents the maximum deviation of the ensemble model predictions from the reference energy. Predictions of the model are shown in Fig. 2.6.

The question then arises: what is the maximum tolerated deviation of the model? The chosen value was

$$\varepsilon_{\text{max}} = 1.5\,\text{kcal/mol}. \tag{2.19}$$

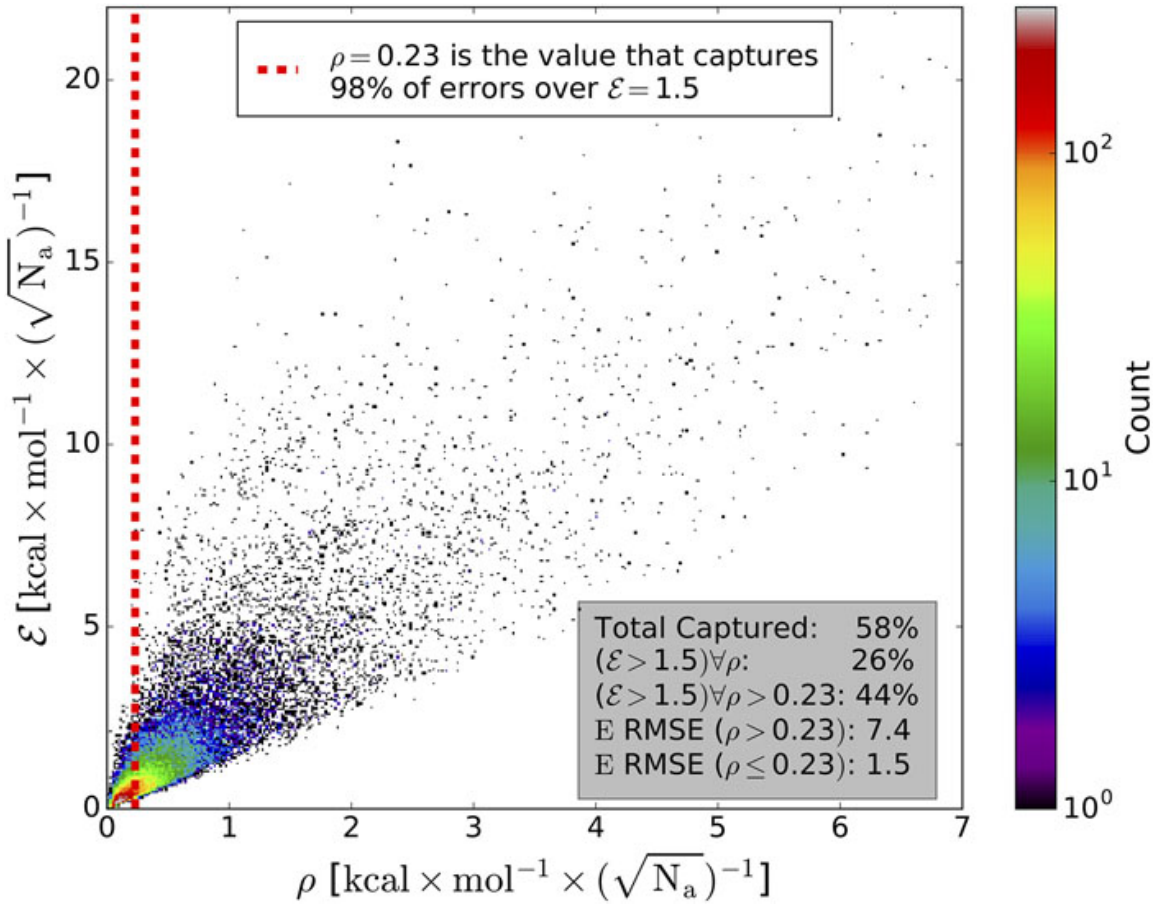

Figure 2.6: Empirical determination of the threshold value, adapted from [15].

The next question is: what percentage of molecules will satisfy Eq. (2.19) for a given $\hat{\rho}$? It was found that for $\hat{\rho} = 0.23$, 98% of the examples satisfying Eq. (2.19) were covered, and this was taken as the threshold. With this threshold, 58% of the examples fell above the allowed value and were included in the new training set.

The final step was to optimize these selected systems to their ground state and then expand the training set again by perturbing them. The methods used were diverse normal mode sampling (DNMS), random trajectory sampling (RTS), and molecular dynamics generated dimer sampling (MDDS). More details can be found in the original paper [15]. For the calculation of energies of these conformations, the ωB97x functional [13] with the 6-31G(d) basis set [14] was again employed.

This cycle was repeated as long as the predictive power of the model continued to improve. The final version obtained through this method was named ANI-1x. A schematic of the process is shown in Fig. 2.7. Despite using only one quarter of the data compared to ANI-1, the network demonstrated a significant improvement in predictive accuracy over its predecessor.

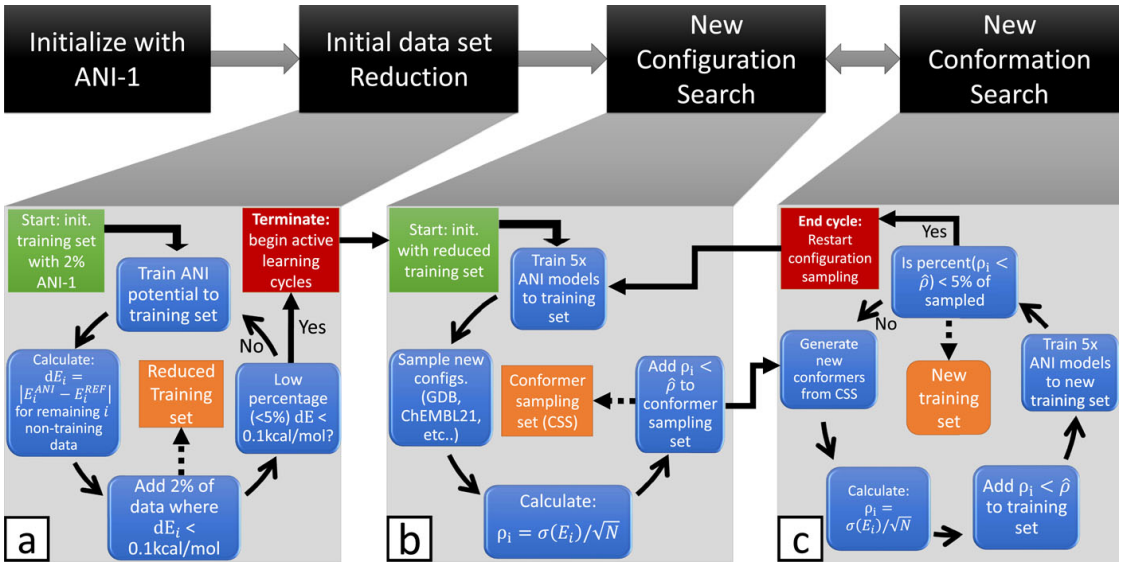

Figure 2.7: Schematic representation of the ANI-1x neural network, adapted from [15].

2.4 ANI-1ccx

The coupled cluster (CC) theory provides a systematic approach to the exact solution of the Schrödinger equation and is considered the gold standard for many applications [20–22]. By employing coupled clusters with single, double, and perturbative triple excitations (CCSD(T)) and extrapolating to the complete basis set (CBS) limit [23, 24], it is possible to accurately describe even the most complex atomic interactions in molecules [25]. Unfortunately, calculations at the CCSD(T)/CBS level are computationally expensive and often impractical for systems with more than a dozen atoms. The idea is therefore to develop a model capable of matching the accuracy of the CC method, while at the same time being widely applicable to other systems. These requirements were met using transfer learning, and the resulting model was named ANI-1ccx [26]. The idea was to first train the model on a larger dataset and then refine it through fine adjustments on the target dataset. A schematic of the model is shown in Fig. 2.8.

The first step was to train the model following the ANI-1x approach, using the previously obtained 5 million training examples, and then refine it on a smaller dataset of half a million entries with more accurately computed energies. Using selected approximations, molecular energies were calculated with accuracy comparable to the CCSD(T)/CBS method, but at a significantly lower computational cost. More details about this procedure can be found in the original paper [26].

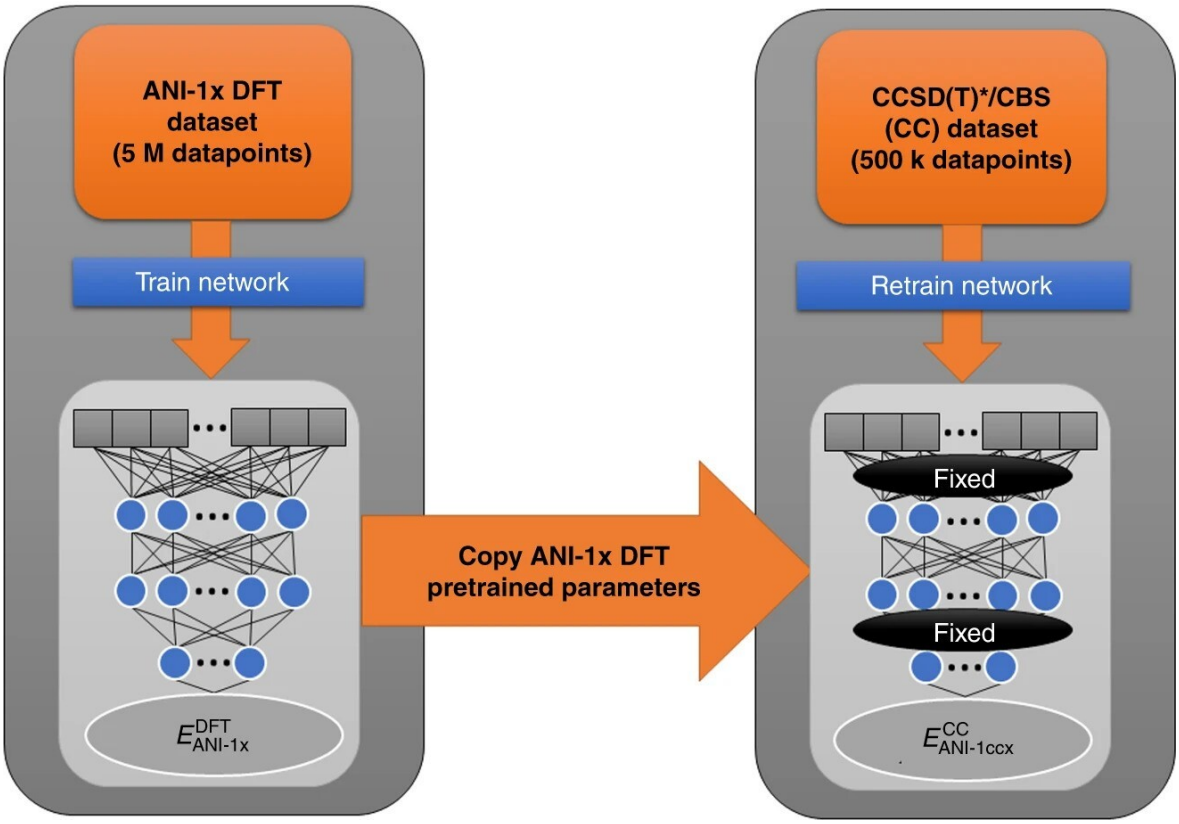

Figure 2.8: Schematic representation of the ANI-1ccx neural network, adapted from [26].

Once all the components for transfer learning were in place, the next step was to further "fine-tune" the existing ANI-1x model on the new examples. Two hidden layers of the network were kept fixed, while the other two were allowed to change. More precisely, of the 325,248 free parameters of the initial model, 65,280 were set as fixed to avoid overfitting.

Figure 2.9 shows the comparison of ANI-1ccx performance with density functional theory (DFT, used in ANI-1 and ANI-1x) on conformations from GDB-10 to GDB-13, with up to 100 kcal/mol deviations from the energetic minimum. The distribution of energy deviations for ANI-1ccx has a standard deviation of 2.3 kcal/mol, while DFT has a deviation of 6.3 kcal/mol. Moreover, the ANI-1ccx distribution is centered symmetrically around zero. DFT, on the other hand, tends to systematically overestimate the system energy, on average by about 15 kcal/mol. In addition to energies, ANI-1ccx was also used for computing forces, reaction and isomerization energies, and molecular torsions, although these applications fall outside the scope of this work.

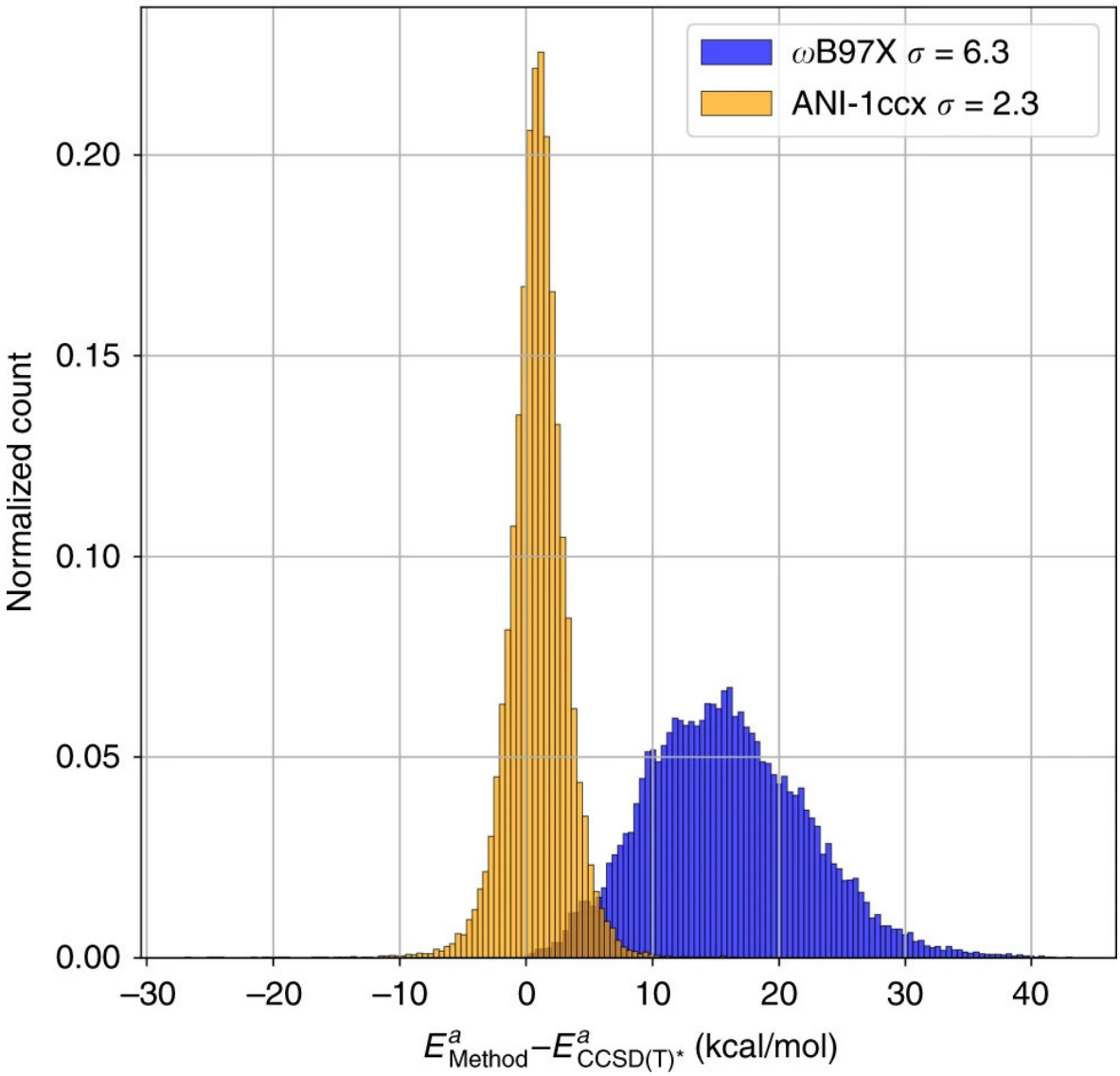

Figure 2.9: Comparison of energy predictions by two models, adapted from [26].

2.5 *E(n) Equivariant Graph Neural Networks*

As the name suggests, the approach of these neural networks is based on equivariance, more precisely on equivariance to rotations, translations, reflections, and graph permutations—properties that naturally hold in molecular systems. The name of the model derives from the group of symmetries of 3D translations and rotations, denoted in group theory as SE(3). When reflections are also included, the notation becomes E(3). Since this model can be applied in higher dimensions, it is given the general notation E(n). An implementation of this model was presented by Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling in 2021 [27]. This implementation was simpler than other approaches with similar goals, as it avoided the use

of spherical harmonics. Moreover, as mentioned earlier, it was not limited to three-dimensional space, and the scaling of complexity with dimension was not drastic.

Let us define equivariance more precisely. Let $T_g : X \to X$ be a set of transformations for an abstract group $g \in G$. A function $\phi : X \to Y$ is said to be equivariant with respect to g if there exists an equivalent transformation on the codomain $S_g : Y \to Y$ such that

$$\phi\big(T_g(x)\big) = S_g\big(\phi(x)\big). \tag{2.20}$$

Consider $\mathbf{x}$ as an n-dimensional vector. For the three properties mentioned earlier, the following relations hold:

- Translation of $\mathbf{x}$ by some $g \in \mathbb{R}^n$ results in the same translation in the codomain: $\phi(\mathbf{x}) + g = \phi(\mathbf{x} + g)$.
- Rotation (and reflection) by a matrix $Q \in \mathbb{R}^{n \times n}$ satisfies $Q(\phi(\mathbf{x})) = \phi(Q\mathbf{x})$, illustrated schematically in Fig. 2.10.
- Permutation of the domain vector followed by mapping yields the same result as mapping first and then permuting in the codomain: $P(\phi(\mathbf{x})) = \phi(P(\mathbf{x}))$.

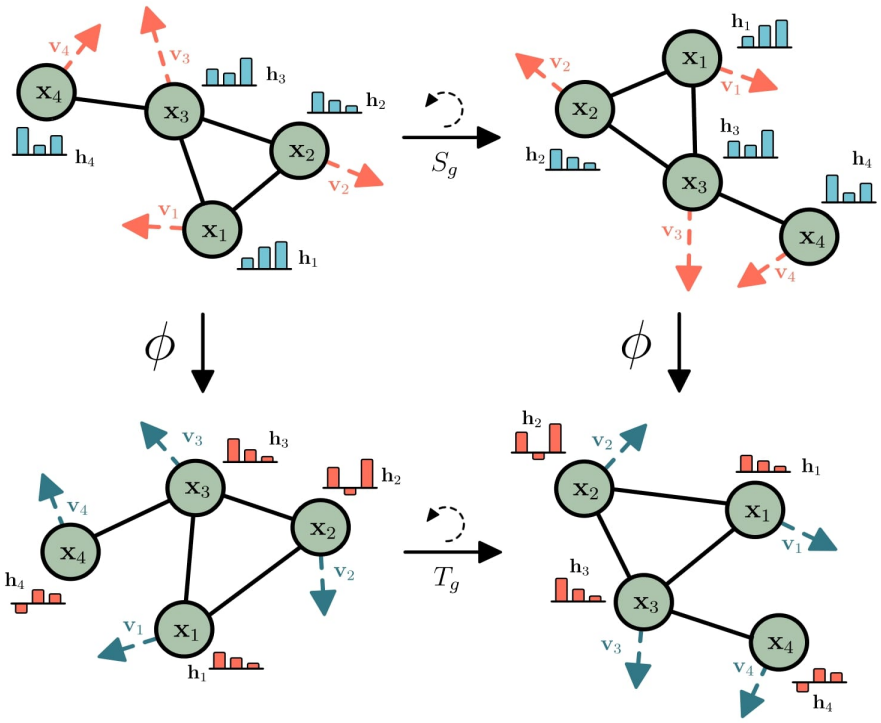

Figure 2.10: Schematic representation of rotational equivariance, adapted from [27].

Typical graph neural networks (GNNs) operate as follows. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with nodes $v_i \in \mathcal{V}$ and edges $\varepsilon_{ij} \in \mathcal{E}$. A convolutional layer on this graph can be defined as

$$\mathbf{m}_{ij} = \phi_e\big(\mathbf{h}_i^l,\, \mathbf{h}_j^l,\, a_{ij}\big) \tag{2.21}$$

$$\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij} \tag{2.22}$$

$$\mathbf{h}_i^{l+1} = \phi_h\big(\mathbf{h}_i^l,\, \mathbf{m}_i\big), \tag{2.23}$$

where $\mathbf{h}_i^l \in \mathbb{R}^k$ is the k-dimensional representation of node v_i in layer l. The parameters a_{ij} are edge attributes, and $\mathcal{N}(i)$ denotes the set of neighbors of node v_i. Finally, ϕ_e and ϕ_h are operations on edges and nodes, respectively, typically approximated by multilayer perceptrons (MLPs).

Now let us consider how equivariant graph neural networks (EGNNs) differ from GNNs. Using the same notation, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a graph with nodes $v_i \in \mathcal{V}$ and edges $\varepsilon_{ij} \in \mathcal{E}$. In addition to node representations $\mathbf{h}_i \in \mathbb{R}^k$, each node is now associated with n-dimensional coordinates $\mathbf{x}_i \in \mathbb{R}^n$. Suppose the graph has M nodes, i.e., $i \in \{1, 2, ..., M\}$. The model preserves equivariance to rotations and translations on the coordinate set $\mathbf{x}_i$, while equivariance to permutations on the set of nodes $\mathcal{V}$ is also maintained, as in GNNs. An EGNN layer is defined by the following equations:

$$\mathbf{m}_{ij} = \phi_e\Big(\mathbf{h}_i^l,\, \mathbf{h}_j^l,\, \big|\big|\mathbf{x}_i^l - \mathbf{x}_j^l\big|\big|^2,\, a_{ij}\Big) \tag{2.24}$$

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + C \sum_{j \neq i} \big(\mathbf{x}_i^l - \mathbf{x}_j^l\big)\phi_x(\mathbf{m}_{ij}) \tag{2.25}$$

$$\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij} \tag{2.26}$$

$$\mathbf{h}_i^{l+1} = \phi_h\big(\mathbf{h}_i^l,\, \mathbf{m}_i\big). \tag{2.27}$$

The edge operation is modified by including the squared relative distance between two coordinates, $\big|\big|\mathbf{x}_i^l - \mathbf{x}_j^l\big|\big|^2$, yielding Eq. 2.24. The input parameters—node representations $\mathbf{h}_i^l$, $\mathbf{h}_j^l$, and edge attributes a_{ij}—remain unchanged. For this work, edge attributes will include edge values $a_{ij} = \varepsilon_{ij}$, though additional information can

also be incorporated.

Equation (2.25) represents the update of the position vector of each node. As shown, the position is updated by a weighted sum of all relative distances $(\mathbf{x}_i - \mathbf{x}_j)$ for each j. The weights are computed by the function $\phi_x : \mathbb{R}^k \to \mathbb{R}$, which takes the edge representation $\mathbf{m}_{ij}$ from the previous step as input and returns a scalar. The constant C is chosen as $1/(M-1)$, dividing the sum by its number of terms. This expression is the key difference between EGNNs and GNNs and the reason why translational and rotational equivariances are preserved (the proof is given in the original paper [27]). Despite its simplicity, this equivariant operation is highly flexible, as the representation $\mathbf{m}_{ij}$ can now carry information from the entire graph, not only from the given edge ε_{ij}. Due to these invariances, EGNNs perform better in modeling three-dimensional problems compared to other graph-based neural networks.

Equations (2.26) and (2.27) follow the same update rules as standard GNNs. Equation (2.26) aggregates all incoming information from neighboring nodes $\mathcal{N}(i)$ with respect to node v_i, while Eq. 2.27 performs the update at node v_i by taking as input the aggregated message $\mathbf{m}_i$ and the node representation $\mathbf{h}_i^l$, producing the updated representation $\mathbf{h}_i^{l+1}$.

It should be noted that Eq. 2.26 is only one of several possible aggregation schemes. Common alternatives to summation include:

- **Mean:** Eq. 2.26 is divided by the number of neighbors $\mathcal{N}(i)$, effectively taking the average. This normalizes the contribution of each neighbor, preventing nodes with many neighbors from exerting disproportionate influence. This is useful when node degrees vary significantly.

- **Maximum:** Instead of summation, the maximum value of $\mathbf{m}_{ij}$ among neighbors is taken, and the updated value equals it. This allows the model to focus on the most salient neighbor features, useful when a single dominant neighbor determines the outcome.

- **Minimum:** Similar to maximum aggregation, but using the minimum value. Though less common, this can be useful when the smallest influence or worst-case scenario is most relevant.

- **Weighted sums:** Various forms of weighted summations of $\mathbf{m}_{ij}$, where weights c_{ij} may depend not only on $\mathbf{m}_{ij}$ but also directly on the node representations

$\mathbf{h}_i$ and $\mathbf{h}_j$. This approach is useful when some neighbors should be emphasized more than others, or when the model needs to learn adaptive weighting.

2.6 Random Fourier Features

The authors of [28] used recent advances in modeling the behavior of deep networks through kernel regression and the neural tangent kernel (NTK) [29] to theoretically and experimentally demonstrate that standard multilayer perceptrons (MLPs) are not well suited for low-dimensional tasks in computer vision and graphics based on coordinates. In particular, MLPs struggle to learn high-frequency functions, a phenomenon known in the literature as spectral bias [30, 31]. NTK theory suggests that this is because coordinate-based MLPs correspond to kernels with rapidly decaying frequency spectra, which limits their ability to represent high-frequency content.

Several recent works [32,33] have experimentally shown that heuristic sinusoidal mappings of input coordinates (known as positional encoding) enable MLPs to represent higher-frequency content. The input coordinates $\mathbf{v}$ are mapped into

$$\gamma(\mathbf{v}) = \left[a_1 \cos(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}),\ a_1 \sin(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}),\ ...,\ a_m \cos(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v}),\ a_m \sin(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v})\right]^{\mathrm{T}} \quad (2.28)$$

before being fed into the MLP. It has been shown that this mapping transforms the NTK into a stationary (translation-invariant) kernel and allows the spectrum of the NTK to be tuned by modifying the frequency vectors $\mathbf{b}_j$, thereby controlling the range of frequencies that the corresponding MLP can learn. A simple strategy of setting $a_j = 1$ and sampling $\mathbf{b}_j$ randomly from an isotropic distribution achieves good performance, with the scale (standard deviation) of the distribution having a much greater impact on results than its specific form. In [28], MLPs trained with this Fourier feature mapping on a variety of tasks relevant to computer vision and graphics exhibited dramatically improved performance compared to standard coordinate-based MLPs.

2.7 Final Model and Approach Used in this Work

The hypothesis of this work is that by combining two datasets of different accuracy but similar structures, we can obtain a better model than by using only one of them. The two datasets used are ANI-1x and ANI-1ccx, with the latter providing more accurate energy values. We have approximately 4.6 million molecule–energy pairs from ANI-1x and 500,000 from ANI-1ccx. Ideally, training would involve all available data; however, due to computational constraints and limited resources, smaller subsets were selected and partitioned into training, validation, and test sets as needed. Details on this partitioning are provided in the Results and Discussion section.

Since both ANI-1x and ANI-1ccx datasets consist of only four elements (H, C, N, O), it would in principle be sufficient to encode each element as a scalar from the set $\{1, 6, 7, 8\}$. While sufficient, this representation is far from optimal. Instead, in this work each element was represented by a vector orthogonal to the others. Because four mutually orthogonal vectors are required, their minimal dimension is also four. This transformation is known in practice as one-hot encoding. The vectors assigned to the elements are:

$$\mathrm{H} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathrm{C} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \mathrm{N} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad \mathrm{O} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{2.29}$$

Although the difference may seem minor, this representation enables the neural network to more clearly distinguish between elements and increases the number of free parameters, thereby improving predictive power.

It is also useful for the network to distinguish data originating from different datasets. Instead of defining a global variable for each molecule (e.g., 1 for ANI-1x and 0 for ANI-1ccx), this distinction can be achieved by manually adding a fifth dimension to the atomic vectors. For ANI-1x, the representation becomes:

$$\mathrm{H} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \mathrm{C} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \mathrm{N} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \qquad \mathrm{O} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \tag{2.30}$$

and for ANI-1ccx:

$$\mathrm{H} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathrm{C} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathrm{N} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \mathrm{O} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}. \tag{2.31}$$

In this way, we create the equivalent of a global variable without implementing it separately. The network can now distinguish molecules from the two datasets. It should be noted that the orthogonality of the ANI-1x vectors is broken by adding the fifth dimension. For distinguishing atoms within a molecule, strict orthogonality of vectors is not required, as shown in the initial scalar representation. Although orthogonality is desirable, it is not essential and does not hinder recognition of atoms.

To improve results, it is clear that a standard graph neural network (GNN) is insufficient, and a different approach is required. For this reason, an equivariant graph neural network (EGNN) was used in combination with random Fourier features (RFF). As discussed in Section 2.5, the EGNN approach enables interactions between nodes that are not directly connected. The depth of these interactions can be chosen—effectively answering the question: which order of neighbors will be included in the interaction? This is regulated by the number of EGNN layers applied in each epoch of training. In the first cycle, each atom is updated with information from its neighbors. In the second cycle, each atom receives information from neighbors that now already include information from their neighbors—effectively allowing the original atom to interact with second-order neighbors, and so forth. Increasing the number of layers improves predictive power, and ideally, each atom would be updated with information from all others, though this comes at high computational cost. Empirically, a good balance between predictive accuracy and computational

efficiency is achieved with three layers. Since atomic coordinates are updated with each layer, it is not sufficient to provide only the molecular graph structure; exact atomic positions are required. Information aggregation at the node level was performed using summation, which is appropriate here as we require the cumulative influence of all neighbors.

For representing interatomic distances, RFFs were employed. This approach provides a scalable and flexible means of incorporating nonlinear relationships between node distances. Unlike directly mapping distances to Gaussian features, which offers a fixed and limited transformation, RFFs project input distances into a higher-dimensional space. This improves expressivity, enabling the network to model more complex interactions between nodes. Moreover, RFFs allow better generalization by approximating a broader spectrum of functions, making them particularly useful in capturing complex patterns in data. Their scalability and adjustable dimensionality make RFFs a valuable tool for enhancing GNN performance. In this work, the scaling factor a_j (see Section 2.6) was set to 1, and the distance representation dimension was chosen as 64.

It is also worth noting the use of pooling. After information passes through the EGNN layers, each node-level feature (representing an atom) contains information from neighboring nodes. However, in tasks such as molecular property prediction, including energy prediction, the goal is to obtain a single output for the entire graph (molecule) rather than separate outputs for each node. The process of combining these node-level features is called pooling. Several pooling methods exist, and in essence, they follow the same principle as aggregation in EGNN layers (Section 2.6). Possible approaches include summation, mean, maximum, minimum, or weighted sums. In this work, simple summation was chosen, expressed as

$$\mathbf{h}_g = \sum_{i \in \mathcal{V}} \mathbf{h}_i, \tag{2.32}$$

where $\mathbf{h}_g$ denotes the graph-level representation, $\mathbf{h}_i$ the node representation, and $\mathcal{V}$ the set of all nodes in the graph. Since energy is an additive property of the system, i.e., the total energy is the sum of contributions of all atoms, summation pooling is best suited to our needs. Another key advantage of this approach is invariance to the order of nodes in the graph. Thus, regardless of node indexing, the sum—and hence

the predicted energy—remains the same.

As a preprocessing step, the atomic contributions to the total energy were subtracted. This helps to isolate binding energies from inherent atomic energies, ensuring comparability across training sets. It also enables a more precise analysis of interatomic interactions without the influence of intrinsic atomic contributions. The Python script used for this procedure, along with all other code for training the neural network, is provided in the Appendix.

Such a model could also be applied to crystals. For instance, by combining one of the ANI datasets with molecular crystals (composed only of H, C, N, O), their properties could be predicted more accurately. EGNNs would again provide long-range atomic interactions, while the periodicity of RFFs would be particularly advantageous due to the periodic structure of crystals.

3 Results and Discussion

3.1 Data Distribution

The model used in this work is described in Section 2.7 and is based on a combination of EGNN with RFF, with some additional modifications enabling it to process data from different datasets. This approach increases the complexity of the network with the expectation that it may capture subtle details of interatomic interactions and thereby improve upon the existing model. As mentioned earlier, a combination of molecules from the ANI-1x and ANI-1ccx datasets was used, and a total of eight different training runs were performed with the data distribution shown in Table 3.1.

Models	**Training**		**Validation**		**Testing**	
	ANI-1x	**ANI-1ccx**	**ANI-1x**	**ANI-1ccx**	**ANI-1x**	**ANI-1ccx**
ANI-1x	160k	0	20k	0	0	20k
ANI-1ccx	0	20k	0	20k	0	20k
20k/20k	20k	20k	10k	20k	0	20k
80k/20k	80k	20k	10k	20k	0	20k
160k/20k	160k	20k	20k	20k	0	20k
20k/60k	20k	60k	10k	20k	0	20k
80k/60k	80k	60k	10k	20k	0	20k
160k/60k	160k	60k	20k	20k	0	20k

Table 3.1: Data distribution across models.

The goal was to compare the accuracy of the models with respect to the ratio of data from the two datasets. To this end, two groups of models were trained: in one group, the number of ANI-1ccx samples was kept constant while the number of ANI-1x samples was varied. Additionally, for comparison, two baseline models without data mixing were trained: ANI-1x and ANI-1ccx, each using only its respective dataset for training and validation. For all models, the test set consisted exclusively of ANI-1ccx data, since our interest lies in predicting energies on this dataset.

The combination of data in the validation set had minimal impact on performance, since validation was used only to regulate the learning rate. If the mean squared error (MSE) on the validation set did not decrease over 10 consecutive epochs, the learning rate was reduced by a factor of 2. The initial learning rate was set to 10^{-4}, and training terminated once it fell below 10^{-7}.

The specific number of samples chosen from each dataset was arbitrary; however, in most cases the number of ANI-1x samples was greater than or equal to the number of ANI-1ccx samples. This reflects the realistic proportion when considering all available data. Furthermore, when applying the model to other systems, the data intended to adapt the model will almost always be in the minority. The final dataset sizes selected for this study were chosen to balance training speed with sufficient data for problem generalization.

3.2 Results

Let us examine the dependence of the loss function on the number of epochs for training and validation (Figs. 3.1 and 3.2). The chosen loss function was the root of the previously defined mean squared error (MSE). For visualization, Gaussian smoothing was applied using the `gaussian_filter1d` function from the `scipy.ndimage` package [34]. Gaussian smoothing is based on averaging nearby points with weights following a Gaussian distribution. For the training loss function, a value of $\sigma = 2$ was used, and for the validation loss function, $\sigma = 4$. This was done to avoid the jaggedness of the curves caused by oscillations and to improve readability.

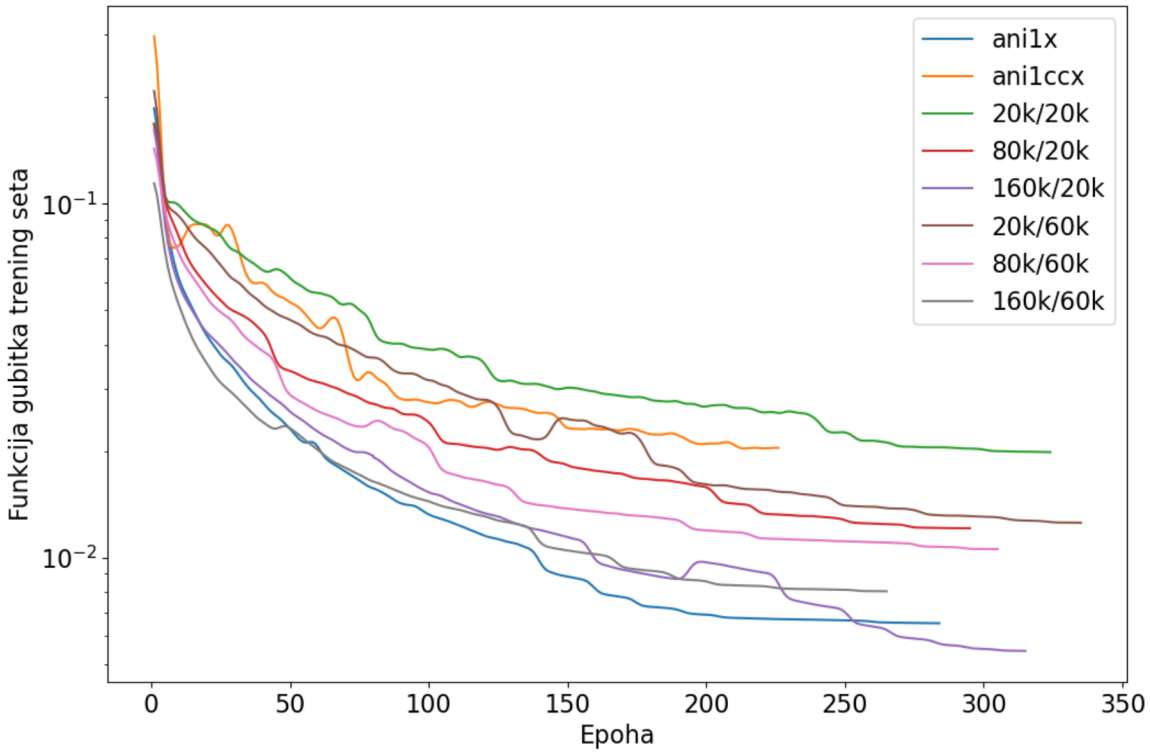

Figure 3.1: Dependence of the loss function on the training set with respect to epoch number for different data combinations.

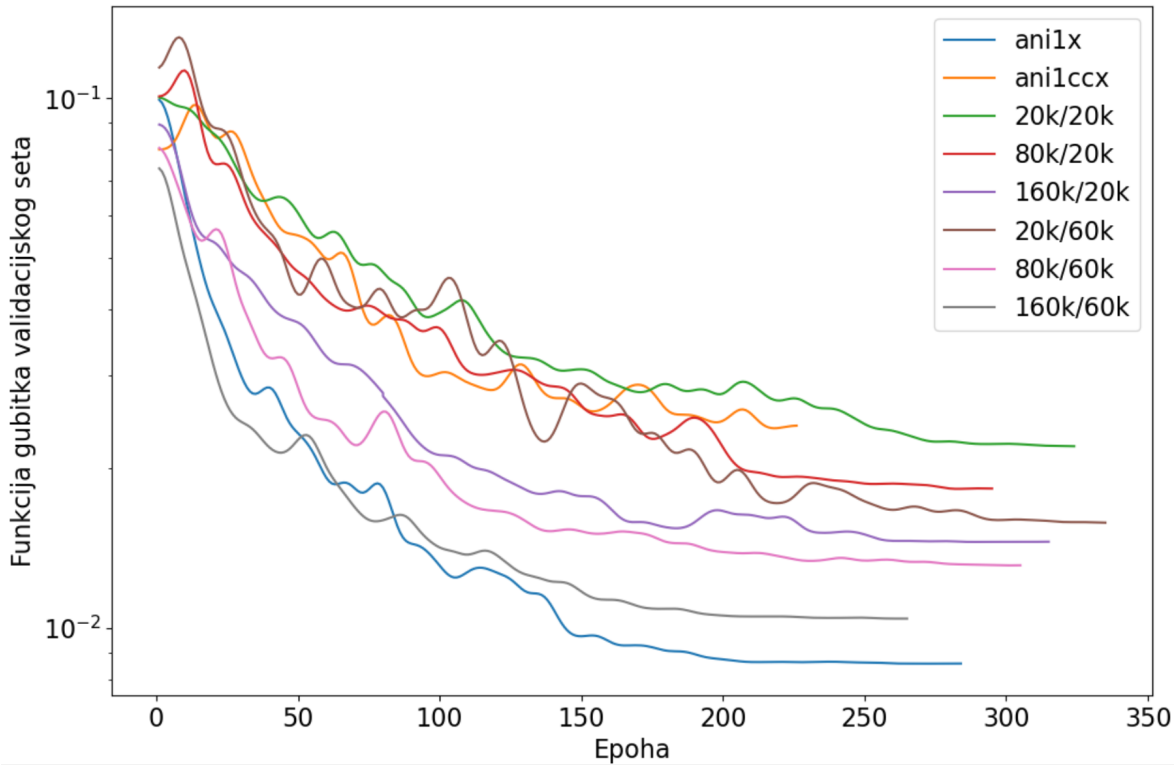

Figure 3.2: Dependence of the loss function on the validation set with respect to epoch number for different data combinations.

Figures 3.1 and 3.2 show the training and validation loss functions for different models. As expected, when the number of ANI-1ccx samples is kept fixed, the values of the training and validation loss functions for mixed datasets behave inversely proportional to the number of ANI-1x samples. What primarily interests us, however, is the comparison of models on the test set. Since all test sets are identical, this provides a direct measure of model accuracy. This comparison is presented in Fig. 3.3.

Here we clearly observe the relative "quality" of the models. Adding a small number of ANI-1x samples made the model slightly worse. However, each subsequent increase in the training set size led to an improvement, with the 80k/20k and 160k/20k models outperforming ANI-1ccx alone. Increasing the number of ANI-1ccx samples in the training set while reducing the number of ANI-1x samples also improved the model. Naturally, this result depends on the scaling factors used for these sets. In cases where the increase is minimal but the reduction drastic, performance would degrade. At higher numbers of ANI-1ccx training samples, the same trend is observed: increasing ANI-1x data reduces the loss further. We conclude that combining the datasets indeed improves the model.

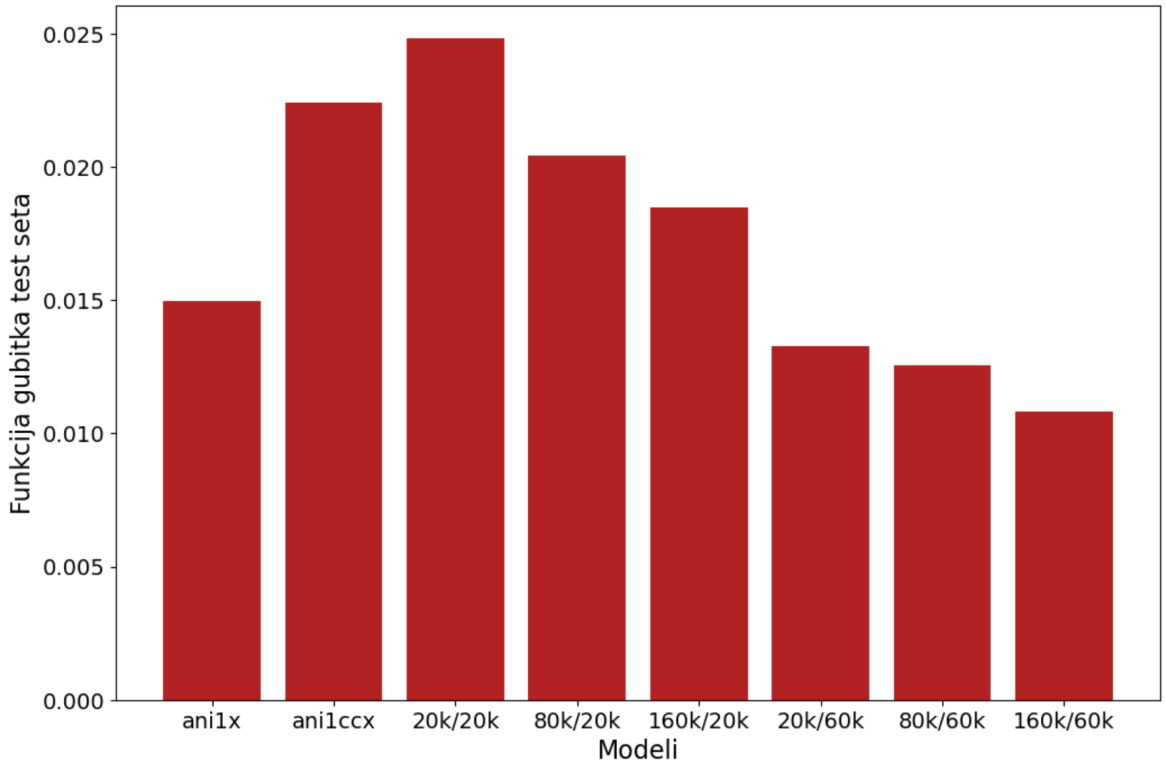

Figure 3.3: Test set loss values for different data combinations.

It is worth commenting on the model trained solely on ANI-1x data. As expected, the training and validation losses are among the lowest compared to other models. The reason is that the model contains a sufficient quantity of a single type of data, enabling the network to "learn" the properties of molecules it is being evaluated on.

At first glance, the test set loss of the ANI-1x-only model compared to others may appear unexpected. Why does adding a smaller amount of ANI-1ccx data make the model worse, even though the test set consists exclusively of ANI-1ccx molecules? The explanation lies in the composition of the ANI-1ccx dataset. Recall that ANI-1x, and thus ANI-1ccx, consists of carefully selected molecules, each carrying unique information useful for network training. If only a portion of these molecules is used for training, the test set will differ significantly, which can negatively impact final performance. As the number of ANI-1ccx training samples increases, this effect diminishes because the network gains access to a larger diversity of molecules, which is evident in the improved performance of the model with 60,000 samples from this dataset.

Finally, it should be noted that the test set loss values remain within acceptable bounds. Typically, neural network errors are around 1 meV/atom. The errors of these models range from 25 to 12 meV per molecule. Considering that the average

molecule in these datasets contains just under 20 atoms, the resulting errors are close to the target value of approximately 1 meV per atom.

3.3 Discussion

Several additional details regarding this process and its generalization remain to be discussed. Let us first examine the relation between training time and dataset size. The sizes of the training sets were in the ratio $2 : 5 : 9$ (for the combined datasets with 20k ANI-1ccx samples), and the required training time per epoch scaled approximately in the same proportion. In other words, model training time scales linearly with the amount of data. The same dependence between time and dataset size was observed for the sets containing 60k ANI-1ccx training samples.

Another issue concerns the choice of test data. Although all models were ultimately tested on the same ANI-1ccx subset, this does not provide a perfect representation of their performance. As mentioned in Section 2.4, the molecules in this dataset were carefully selected to maximize model quality while minimizing dataset size. With this in mind, it is possible that the relative performance of the models would differ slightly if another subset had been chosen. While such changes would not result in drastic differences, it is important to remain aware of this limitation when making conclusions about the generalization of the process.

This type of model also contains many hyperparameters that could be tuned to improve its performance. Although the model produced satisfactory results with the current settings, it is not certain that these are optimal. For example, the number of EGNN layers and the dimensionality of the RFF vectors directly define the network structure. There is also the issue of optimizing the amount of ANI-1x data. Naturally, this depends on whether the primary goal is to maximize predictive performance or to minimize training time. Since we are generally interested in balancing these two aspects, there exists a threshold amount of data beyond which the model no longer provides significant improvements, while becoming computationally more expensive.

Finally, let us comment on potential applications of this model to other systems. Now that we have demonstrated the validity of the approach, the question arises of where else it could be applied. In other words, how different can the combined datasets be while still yielding improvements? As previously mentioned, one promis-

ing system is molecular crystals. Due to their numerous conformations and complex interatomic interactions (van der Waals and hydrogen bonds), the amount of available data for such systems is limited. Since the number of training examples required grows with system complexity, constructing an accurate model based solely on molecular crystals is challenging. A possible solution to this problem could be the combination of molecular crystal data with an ANI-1 dataset.

4 Conclusion

In this work, we employed neural networks based on a combination of the equivariant graph neural network (EGNN) with random Fourier features (RFF). The equivariant approach allows interactions between nodes (representing atoms) that are not directly connected. For our model, the interaction depth was set to include up to third-order neighbors. Representing interatomic distances with Fourier features instead of Gaussian ones provided a stable and flexible way to capture nonlinear relationships. Two datasets were used, ANI-1x and ANI-1ccx, which differ in accuracy. The ANI-1x dataset contains approximately four times more samples but with less precise energy values.

The goal of this work was to demonstrate that combining these two datasets yields a model with better performance than one trained solely on the more accurate ANI-1ccx dataset. To this end, eight models were trained: one using only ANI-1x, one using only ANI-1ccx, and six using different amounts of ANI-1x data combined with ANI-1ccx for training and validation. All test sets were identical and consisted exclusively of ANI-1ccx molecules.

Prior to training, the atomic contributions to the total energy were subtracted from all molecules. This step isolated binding energies from inherent atomic energies, ensuring comparability between different training sets. In addition, the network was given the ability to distinguish molecules based on the dataset of origin.

Individually, all models produced satisfactory results. Test set losses ranged from 25 to 12 meV per molecule. Considering that the average molecule in these datasets consists of about 20 atoms, the error amounts to approximately 1 meV per atom, which is consistent with the accuracy of current state-of-the-art models. Two groups of three models were trained, each with a fixed number of ANI-1ccx samples. The first group contained 20k ANI-1ccx samples, with ANI-1x ratios of 1:1, 1:4, and 1:8. The second group contained 60k ANI-1ccx molecules, with ANI-1x ratios of 3:1, 3:4, and 3:8. In all models using combined datasets, increasing the number of ANI-1x samples in the training set improved the predictive power of the network. Although the first combination performed worse than the ANI-1ccx-only model, all subsequent combinations outperformed it.

While further testing would be required to fully optimize the model, we have

demonstrated that combining datasets of different accuracy levels indeed improves performance. The question remains as to where else such an approach could be beneficial. One promising application is in molecular crystals. Their complexity makes them challenging to model, and the amount of available data for these systems is limited. By combining molecular crystal data with ANI-1 datasets, it may be possible to overcome the lack of sufficient training data for neural networks trained exclusively on molecular crystals.

Appendix

A link to the GitHub repository containing all scripts used in the preparation of this thesis is provided below. The repository includes a `README.md` file with further details about the individual scripts.

`https://github.com/FranK1308/Diplomski`

References

[1] Dronskowski, R. Computational Chemistry of Solid State Materials. 1st ed. : Wiley-VCH, 2005.

[2] Vink, R. L. C.; Barkema, G. T.; Stijnman, M. A.; Bisseling, R. H. Physics Review B : porpose-led publishing, 2001.

[3] Hobday, S.; Smith, R.; Belbruno, J. Applications of neural networks to fitting interatomic potential functions, Vol 7, Number 3 : porpose-led publishing, 1999.

[4] Behler, J.; Parrinello, M. Generalized neural-network representation of high-dimensional potential-energy surfaces // Physical review letters 98, 2007.

[5] Behler, J. Neural network potential-energy surfaces in chemistry: a tool for large-scale simulations. // Physical Chemistry Chemical Physics. Vol. 13, 40 (2011), str. 17930–17955.

[6] Behler, J.; Martoňák, R.; Donadio, D. et al. Pressure-induced phase transiti ons in silicon studied by neural network-based metadynamics simulations. // physica status solidi (b). Vol. 245, 12 (2008), str. 2618–2629.

[7] Behler, J.; Martoňák, R.; Donadio, D. et al. Metadynamics simulations of the high-pressure phases of silicon employing a high-dimensional neural network potential. // Physical review letters. Vol. 100, 18 (2008), str. 185501.

[8] Smith, J. S.; Isayev, O.; Roitberg, A. E. Ani-1: an extensible neural network potential with dft accuracy at force field computational cost. // Chemical science. Vol. 8, 4 (2017), str. 3192–3203.

[9] Behler, J. Perspective: Machine learning potentials for atomistic simulations. // The Journal of chemical physics. Vol. 145, 17 (2016), str. 170901.

[10] Karlik, B.; Olgac, A. V. Performance analysis of various activation functions in generalized mlp architectures of neural networks. // International Journal of Artificial Intelligence and Expert Systems. Vol. 1, 4 (2011), str. 111–122.

[11] Fink, T.; Raymond, J. L. Virtual Exploration of the Chemical Universe up to 11 Atoms of C, N, O, F: Assembly of 26.4 Million Structures (110.9 Million

Stereoisomers) and Analysis for New Ring Systems, Stereochemistry, Physicochemical Properties, Compound Classes, and Drug Discovery // Journal of Chemical Information and Modeling. Vol 747 2(2007), str. 342–353.

[12] Fink, T.; Bruggesser, H.; Raymond, J. L. Virtual Exploration of the Small-Molecule Chemical Universe below 160 Daltons // Angewandte Chemie International Edition. Vol 44, 10(2005), str. 1504-1508.

[13] Chai, J.;Head-Gordon, M. The Journal of Systematic optimization of long-range corrected hybrid density functionals // The Journal of Chemical Physics. Vol 128, 8(2008)

[14] Ditchfield, R.; Hehre, W. J.; Pople, J. A. Self-Consistent Molecular-Orbital Methods. IX. An Extended Gaussian-Type Basis for Molecular-Orbital Studies of Organic Molecules // The Journal of Chemical Physics. Vol 54, 2(1971)

[15] Smith, J. S.; Nebgen, B.; Lubbers, N. et al. Less is more: Sampling chemical space with active learning. // The Journal of chemical physics. Vol. 148, 24(2018), str. 241733.

[16] Jupp, S.; Malone, J.; Bolleman, J.; Brandizi, M.; Davies, M.; Garcia, L.; Gaulton, A.; Gehant, S.; Laibe, C.; Redaschi, N.; Wimalaratne, S. M.; Martin, M.; Le Novre, N.; Parkinson, H.; Birney, E.; Jenkinson, A. M. The EBI RDF platform: linked open data for the life sciences // Bioinformatics. Vol. 30, 9(2014), str. 1338-1339.

[17] Bento, A. P.; Gaulton, A.; Hersey, A.; Bellis, L. J.; Chambers, J.; Davies, M.; Krüger, F. A.; Light, Y.; Mak, L.; McGlinchey, S.; Nowotka, M.; Papadatos, G.; Santos, R.; Overington, J. P. The ChEMBL bioactivity database: an update // Nucleic Acids Research. Vol 42, D1(2014), str. D1083–D1090.

[18] Davies, M.; Nowotka, M.; Papadatos, G.; Atkinson, F.; van Westen, G.; Dedman, N.; Ochoa, R.; Overington, J. MyChEMBL: A Virtual Platform for Distributing Cheminformatics Tools and Open Data // Challenges 5, 334(2014).

[19] Seung, H. S.; Opper, M.; Sompolinsky, H. Query by committee. Proceedings of the fifth annual workshop on Computational learning theory 1992 str. 287–294.

[20] Purvis, G. D.; Bartlett, R. J. A full coupled-cluster singles and doubles model: the inclusion of disconnected triples // The Journal of Chemical Phyics. 76(1982), str. 1910–1918.

[21] Bartlett, R. J.; Musiał, M. Coupled-cluster theory in quantum chemistry // Reviews of Modern Physics 79(2007), str. 291–352.

[22] Daniel Crawford, T.; F. Schaefer, H. III An introduction to coupled cluster theory for computational chemists // Reviews in Computational Chemistry 14(2007), str. 33–136.

[23] Hobza, P.; Šponer, J. Toward true DNA base-stacking energies: MP2, CCSD(T), and complete basis set calculations // Journal of American Chemical Society 124(2002), str. 11802–11808.

[24] Feller, D.; Peterson, K. A.; Crawford, T. D. Sources of error in electronic structure calculations on small chemical systems // The Journal of Chemical Phyics 124(2006).

[25] Řezáč, J.; Riley, K. E.; Hobza, P. Extensions of the S66 data set: more accurate interaction energies and angular-displaced nonequilibrium geometries // Journal of Chemical Theory and Computation 7, 11(2001), str. 3466–3470.

[26] Smith, J.S.; Nebgen, B.T.; Zubatyuk, R. et al. Approaching coupled cluster accuracy with a general-purpose neural network potential through transfer learning // Nature Communications Vol. 10, 2903(2019).

[27] Satorras, V. G.; Hoogeboom, E.; Welling, M. Proceedings of the 38th International Conference on Machine Learning : E(n) Equivariant Graph Neural Networks // Proceedings of Machine Learning Research

[28] Tancik, M.; Srinivasan, P. P.; Mildenhall, B.; Fridovich-Keil, S.; Raghavan, N.; Singhal, U.; Ramamoorthi, R.; Barron, J. T.; Ng, R. Advances in Neural Information Processing Systems 33 : Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains // Conference on Neural Information Processing Systems (2020)

[29] Jacot, A.; Gabriel, F.; Hongler, C. Neural Tangent Kernel: Convergence and generalization in neural networks // Advances in Neural Information Processing Systems 31, 2018.

[30] Basri, R.; Galun, M.; Geifman, A.; Jacobs, D.; Kasten, Y.; Kritchman, S. Proceedings of the 37th International Conference on Machine Learning: Frequency bias in neural networks for input of non-uniform density // Proceedings of Machine Learning Research (2020)

[31] Rahaman, N.; Baratin, A.; Arpit, D.; Draxler, F.; Lin, M.; Hamprecht, F. A.; Bengio, Y.; Courville, A. On the spectral bias of neural networks // International Conference on Machine Learning, 2019.

[32] Mildenhall, B.; Srinivasan, P. P.; Tancik, M.; Barron, J. T.; Ramamoorthi, R.; Ng, R. NeRF: Representing scenes as neural radiance fields for view synthesis // Communications of the Association for Computing Machinery Vol. 65, Issue 1 (2021)

[33] Zhong, E. D.; Bepler, T.; Davis, J. H.; Berger, B. Reconstructing continuous distributions of 3D protein structure from cryo-EM images // International Conference on Learning Representations (2020)

[34] SciPy library: Multidimensional image processing (ndimage) `https://docs.scipy.org/doc/scipy/tutorial/ndimage.html`