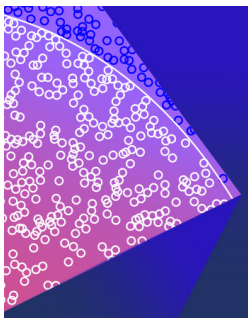
RESEARCH ARTICLE | APRIL 25 2023

ænet-PyTorch: A GPU-supported implementation for machine learning atomic potentials training

Special Collection: Software for Atomistic Machine Learning

Jon López-Zorrilla ; Xabier M. Aretxabaleta ; In Won Yeu ; Iñigo Etxebarria ; Hegoi Manzano ; Nongnuch Artrith

J. Chem. Phys. 158, 164105 (2023)
https://doi.org/10.1063/5.0146803



The Journal of Chemical Physics ARTICLE scitation.org/journal/jcp

ænet-PyTorch: A GPU-supported implementation for machine learning atomic potentials training

Cite as: J. Chem. Phys. 158, 164105 (2023); doi: 10.1063/5.0146803
Submitted: 16 February 2023 • Accepted: 4 April 2023 •
Published Online: 25 April 2023

Jon López-Zorrilla,[1,a)] Xabier M. Aretxabaleta,[1] In Won Yeu,[2] Iñigo Etxebarria,[1,3] Hegoi Manzano,[1] and Nongnuch Artrith[4,b)]

AFFILIATIONS

[1] Physics Department, University of the Basque Country (UPV/EHU), Leioa, Basque Country, Leioa, Spain
[2] Department of Chemical Engineering, Columbia University, New York, New York 10027, USA
[3] EHU Quantum Center, University of the Basque Country (UPV/EHU), Basque Country, Leioa, Spain
[4] Materials Chemistry and Catalysis, Debye Institute for Nanomaterials Science, Utrecht University, Utrecht, The Netherlands

Note: This paper is part of the JCP Special Topic on Software for Atomistic Machine Learning.
[a)] **Electronic mail:** jon.lopezz@ehu.eus
[b)] **Author to whom correspondence should be addressed:** n.artrith@uu.nl

ABSTRACT

In this work, we present ænet-PyTorch, a PyTorch-based implementation for training artificial neural network-based machine learning interatomic potentials. Developed as an extension of the atomic energy network (ænet), ænet-PyTorch provides access to all the tools included in ænet for the application and usage of the potentials. The package has been designed as an alternative to the internal training capabilities of ænet, leveraging the power of graphic processing units to facilitate direct training on forces in addition to energies. This leads to a substantial reduction of the training time by one to two orders of magnitude compared to the central processing unit implementation, enabling direct training on forces for systems beyond small molecules. Here, we demonstrate the main features of ænet-PyTorch and show its performance on open databases. Our results show that training on all the force information within a dataset is not necessary, and including between 10% and 20% of the force information is sufficient to achieve optimally accurate interatomic potentials with the least computational resources.

I. INTRODUCTION

In recent years, machine learning methods have gained popularity as methods to simulate large complex systems due to their ability to predict properties of materials with high accuracy and low computational cost. In particular, machine learning interatomic potentials[1] (MLPs) are data-driven methods that allow the prediction of energies and forces of atomic structures with precision similar to that of the scheme used to generate the reference data but several orders of magnitude faster. Reference data are usually obtained by first-principles calculations, such as density functional theory[2,3] (DFT) for bulk systems or post-Hartree–Fock methods for molecular ones.[4,5] Once trained, these potentials can be employed in conjunction with other advanced simulation techniques, such as molecular dynamics[6] (MD), Monte Carlo[7–9] (MC), and evolutionary algorithms.[10] MLPs have been used with great success in a wide variety of fields including physics, chemistry, and industry, in applications such as computing phonon properties,[11,12] studying phase diagrams,[13,14] or predicting properties and structures of crystals and molecules.[15–18]

Multiple MLP approaches have been proposed in the literature; some examples include artificial neural network-based potentials (ANN-based MLPs),[19–21] Gaussian approximation potentials,[22–24] kernel-based methods,[25–27] message-passing networks,[28–30] or spectral neighbor analysis potentials[31,32] among many others. In this case, our focus lies on the first group, ANN-based MLPs, which is

based on one of the oldest and most studied methods in machine learning. The main drawback of MLPs is that the calculations to compile the reference dataset and training the potential is time- and resource-intensive. Several fits are usually needed, i.e., the training needs to be repeated several times, first to select the optimal hyperparameters and then to refine the MLP with techniques such as active or on-the-fly learning.[33–36] The simplest of the approaches relies on training only on the energies of the reference structures. However, this approach usually leads to poor predictions of forces, which are the negative gradient of the predicted potential energy with respect to the atomic coordinates and are of foremost importance to achieve long stable MD simulations.[37] Therefore, efficient techniques are required to include force information in the training in addition to the reference potential energies, which is far more computationally demanding.

As in most fields concerning machine learning, Graphics Processing Units (GPUs) provide the best solution to this problem. Some work has already been done in recent years in this direction, leading to updates of codes for training MLPs to include GPU support. For instance, that is the case of ANI,[38] AMP,[39] or deepMD,[40] to name but a few. Here, we present an extensive update of the atomic energy network (ænet)[41] code to allow MLP training on GPUs. ænet has proved to be efficient, especially when handling systems with many species. Our approach is simple: by using a well-known ML framework, PyTorch,[42] we replace the training process of ænet while keeping full compatibility with all the other ænet resources. In this paper, we describe the main characteristics of our code called ænet-PyTorch and show its potential to train on both system energy and atomic forces. We also show that training on all the forces of a database is redundant by testing the code on several open databases. The code will be available on GitHub as free, open-source software.

A. Machine learning potentials

The main reason for the success of ANN-based MLPs is that once trained, they can be used to predict energies and forces of new structures independently of the number of atoms, yet they are limited to the chemical species present in the reference data. This is achieved by partitioning the energy of the system into atomic contributions,

$$E^{\mathrm{ANN}}(\{\sigma^{(i)}\}) = \sum_{i=1}^{N_{\mathrm{atom}}} E_i(\sigma^{(i)}). \tag{1}$$

A neural network is trained for each chemical element present in the reference data, and the contribution of each atom (E_i) is then evaluated using the network specific to its element. It is assumed that the contribution of each atom i depends only on its local environment (denoted as $\sigma^{(i)}$ in the previous equation). By numerically describing those environments, via the so-called atomic fingerprints or descriptors and training on the total energy of the system, the resulting potential can be generalized independently of the number of atoms. Several ways of representing atomic environments can be found in the literature,[43–47] all of them satisfying a set of conditions regarding symmetry with respect to the exchange of equivalent atoms, rotations and translations of the structures, and smoothness of the descriptor functions.

The training is an optimization process, so the first step is to define the objective function to be minimized. There are two main approaches: training only on energies or including also information about the forces acting on each atom. In any case, the objective function (also known as the loss function) is usually the root mean squared error (RMSE) of the ANN output compared to the reference value. We define the loss function ($\mathscr{L}_{EF}$) as a weighted sum of both energy and force errors,

$$\mathscr{L}_{EF} = (1-\alpha)\mathscr{L}_E + \alpha\mathscr{L}_F, \tag{2}$$

where α is a free weight parameter. RMSEs for energy ($\mathscr{L}_E$) and forces ($\mathscr{L}_F$) are defined as

$$\mathscr{L}_E = \sqrt{\frac{1}{N_{\mathrm{struc}}}\sum_{i=1}^{N_{\mathrm{struc}}}\left(E_i^{\mathrm{ANN}} - E_i^{\mathrm{REF}}\right)^2} \tag{3}$$

and

$$\mathscr{L}_F = \sqrt{\frac{1}{\sum_{i=1}^{N_{\mathrm{struc}}} 3N_{\mathrm{atom},i}}\sum_{i=1}^{N_{\mathrm{struc}}}\sum_{j=1}^{N_{\mathrm{atom},i}}\left(\mathbf{F}_{i,j}^{\mathrm{ANN}} - \mathbf{F}_{i,j}^{\mathrm{REF}}\right)^2}, \tag{4}$$

where N_{struc} is the number of structures in the database and $N_{\mathrm{atom},i}$ is the amount of atoms in the ith structure. Here, the superscripts ANN and REF denote the output of the MLP and the reference value, respectively. For the sake of simplicity, the force error is normalized per atom in the whole set, instead of having the average of force error per structure. That is to say, we consider each force vector as an independent training example.[48] The outputs of the MLP ANNs are the energy contributions of each atom in Eq. (1), and the forces acting on each atom can be computed by taking the gradient of the total energy with respect to the coordinates of the atoms ($\{\mathbf{R}_k\}$),

$$\mathbf{F}_k^{\mathrm{ANN}} = -\nabla_k E^{\mathrm{ANN}}(\{\sigma^{(i)}\}) = -\sum_{i=1}^{N_{\mathrm{atom}}}\sum_{n=1}^{N_{\mathrm{descr}}}\frac{\partial E_i}{\partial \sigma_n^{(i)}}\frac{\partial \sigma_n^{(i)}}{\partial \mathbf{R}_k}. \tag{5}$$

Training the network means finding an optimal set of weights and biases that minimize the loss function. It is a common practice to include one more term in the loss function, which is called regularization or weight decay. It is intended to avoid overfitting, i.e., finding a solution that minimizes the error for the training examples but does not generalize to examples outside the reference dataset.[49] Here, we consider the L2 regularization,[50] introduced in the training as an extra term in the loss function,

$$\mathscr{L} = \mathscr{L}_{EF} + \frac{1}{2}\lambda\sum_i w_i^2, \tag{6}$$

where $\{w_i\}$ is the set of all parameters to be fitted during training and λ is a weighting parameter. In the following, that λ parameter will be referred to as the regularization or weight decay term.

In practice, in each training epoch, the loss function in Eq. (6) is not evaluated for the whole set of training examples but rather for a subgroup of them at a time. This group, or batch, is used to approximate the gradient of the loss function and to update the parameters in the network in each epoch.

II. ænet-PyTorch

A. ænet

ænet[41] provides a tool for generating, testing, and applying machine learning interatomic potentials based on the Behler–Parrinello method,[1] entirely written in Fortran 2003. Currently, it includes two different descriptors: the original Behler atom-centered symmetry functions[1,51] and the Chebyshev descriptors[52] by Artrith *et al.* Nonetheless, it is worth noting that this update of the code is independent of the descriptor, so any future addition of new descriptors in the original ænet code would be compatible with ænet-PyTorch.

ænet also includes the utilities to employ the MLPs in real applications, for example, in molecular dynamics simulations[6] [using the interface with Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS)[53] or TINKER[54]], in path integral molecular dynamics,[55] or in any of the capabilities provided in the Atomic Simulation Environment[56] Python package. Our new program has been designed as an extension to the original ænet code, and all of these utilities that ænet provides can be readily used with the potentials generated by ænet-PyTorch.

Including force information in the training increases the transferability of the potentials, improving the force prediction and finally enhancing the stability of MD simulations. Nevertheless, despite ænet's parallel implementation's efficient scaling on central processing units (CPUs), training on both energies and forces has been limited to simple systems with a small number of atoms due to its significant requirements for computational time and memory. That is the main reason why we consider this GPU implementation through PyTorch necessary so that the force training becomes feasible for complex systems within our ænet framework.

B. ænet-PyTorch implementation

The principal novelties included in the training scheme of ænet-PyTorch are the capability to train MLPs using GPU in addition to CPU cores and training both on reference energies and atomic forces. ænet-PyTorch also provides users with easy access to various optimization algorithms and overfitting prevention techniques, such as dropout and batch-normalization layers, all available within the PyTorch framework. All of these are easily implementable and customizable for users.

PyTorch is a free and open-source machine learning framework based on the TorchLib library with a Python interface. It contains routines to enable efficient training of deep neural networks with CPU and GPU support, via the C++ and CUDA-based code, respectively. In contrast to other PyTorch-based implementations of MLPs, which usually rely on custom CUDA or C++ modules, this one is written completely in Python using only PyTorch's built-in functions. More specifically, GPUs are best suited for tensor operations, so most operations must be expressed as such to achieve optimal performance.

The optimization strategy (depicted in Fig. 1) involves grouping all atoms of the same species together before training. This allows a single network call to predict the energies of all atoms of that type. PyTorch handles the parallel computation of atomic energies and forces internally. The process of grouping all the atoms and later reordering the resulting atomic energies and forces is also performed using only PyTorch's built-in routines, so it is also a parallel process. As we will see in Sec. III, this idea leads to great scaling of the code, particularly on GPU.

For a detailed explanation of the capabilities and options of the code, the user is referred to the documentation of the

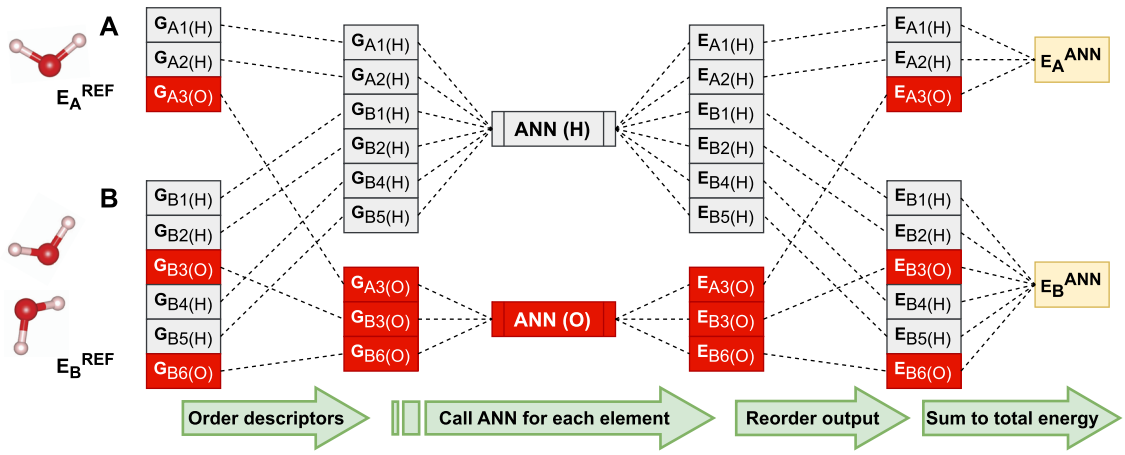


FIG. 1. In this example of the workflow of the code, the dataset is composed of two structures: A with one water molecule and B with a cluster of two water molecules. First, all the descriptors of the hydrogen atoms of both A and B structures are grouped in the same tensor and so are the ones for the oxygen atoms. A single call to the neural network for the hydrogen (oxygen) atoms computes the atomic contribution of all of the atoms of that element. Then, the ordering process is reverted and the atomic energies are grouped per structure again. Finally, the contributions of each atom are summed to obtain the total energy of each structure, E_A^{ANN} and E_B^{ANN}. All these operations are implemented via built-in PyTorch routines.

code, which can be found online in the GitHub repository (https://github.com/atomisticnet/aenet-pytorch) together with the code and an example of its usage. However, some of them are worth noting, and they will be tested in the following. One of the main concerns when developing MLPs is the resources needed to train forces, both in terms of computational time and memory. It must be noted that with our implementation, we decided to sacrifice memory resources to speed up the training several orders of magnitude. However, some options have been included to reduce those memory requirements for large datasets, both CPU and GPU memory (called RAM and VRAM, respectively):

- **GPU (gpu)**: This mode stores all the data needed on the GPU memory. Despite being the fastest way, storing all the data in the GPU device might be problematic for large datasets. The aim of this storage mode is to minimize GPU VRAM usage while optimizing the execution time, assuming that the VRAM is generally the bottleneck. Therefore, all information is loaded into the CPU RAM first and then moved to the VRAM after preprocessing the data.
- **GPU (cpu)**: If enough RAM is available but that of the GPU is limited, the data can be all stored in the CPU RAM. When each batch is processed, only the required data for that batch is moved to the GPU. This one is a slightly slower approach, as it requires constant communication between GPU and CPU.
- **CPU**: All calculations are done using CPU cores, without GPU support.

All three approaches [GPU (gpu), GPU (cpu), and CPU] will be tested in Secs. III A and III B.

III. ænet-PyTorch SCALING

In order to test the performance of ænet-PyTorch, we will replicate already published calculations using this new code and compare the results with those published by their original authors. Let us first consider the database used in the first ænet code release,[41] formed by 7815 structures belonging to different phases of bulk titanium dioxides (TiO_2). In this section, we will use this set of calculations to check the performance of the code for energy and force training. All MLP training would usually require the selection of an architecture for the ANN, but in our case, we will be using the one that the authors proposed and proved to be optimal: 48–15–15–1 architecture using the Chebyshev polynomials as descriptors.

A. Energy training

As we have already stated, this implementation allows the usage of all the tools in PyTorch, starting with all the optimization algorithms. Therefore, we first aim to select the most appropriate set of hyperparameters for the training. This includes the optimization method, batch size, learning rate, and weight decay. To do so, we consider all the variants of the Adam optimizer[57–59] available in PyTorch, batch sizes ranging from 64 to 1024, learning rates from 10^{-6} to 10^{-1}, and weight decays from 10^{-5} to 10^{-2}. This results in a total of 600 training processes.

TABLE I. List of parameters yielding the best training results for the TiO_2 database using only energy information: batch size (BS), learning rate (LR), weight decay (WD), and root mean squared error of the energy of the validation set (RMSE). The RMSE is given in meV/atom.

Method	BS	LR	WD	RMSE
Adagrad	64	10^{-1}	10^{-4}	4.221
Adamw	128	10^{-4}	10^{-5}	4.434
Adamw	256	10^{-4}	10^{-5}	4.387
Adamax	512	10^{-3}	10^{-5}	4.310
Adamax	1024	10^{-3}	10^{-4}	4.877

The models are evaluated based on their accuracy on the independent validation set, and the best results are displayed in Table I. All calculations are performed for 10^4 training epochs to ensure that those with lower learning rates have also converged. In the following, a training epoch refers to a single iteration of the learning algorithm over the entire training dataset, i.e., processing the entire set once, independently of the number of batches in which the data are divided.

It has already been discussed that the feature that makes ænet-PyTorch efficient is the grouping of atoms of the same species before training. This has some implications: the time needed per epoch considerably decreases with the increase of the batch size, but it comes at the cost of more memory. Figure 2 shows the time needed per epoch for different batch sizes. The number of epochs needed to achieve convergence depends on each specific case; notwithstanding, the time per epoch is a good indicator of the scaling of the code. For each value, the best combination of method, learning rate, and regularization have been selected based on the results shown in Table I. The time needed for training considerably decreases when using GPU. The improvement ranges from one to two orders of magnitude, increasing with the batch size. On the other hand, the memory used also increases with the batch size, so in some cases, it would be better to save the dataset information in the CPU RAM so as to find a trade-off between time- and memory-efficiency. In that case, the speedup is still considerable while keeping the GPU memory (which is usually the limiting computational feature) more moderate. However, these problems rarely arise from calculations involving only energies.

These results show that energy training with ænet-PyTorch is efficient even on CPUs, considering that the CPU computations here are performed using only 2 CPU cores. As a reference, this training time is similar to that needed in the original ænet to obtain a similar error training with 16 processors. This comparison is not completely fair, since ænet uses an optimization algorithm much slower, the limited memory BGFS algorithm.

B. Force training

Let us now consider the new feature introduced with this code, i.e., training on atomic forces in addition to energies. In this case, we will use the optimal training hyperparameters selected for energy training, and we will focus on two other parameters influencing force training: the α parameter from Eq. (2) weighting the energy and force RMSEs in the loss function and the fraction of structures

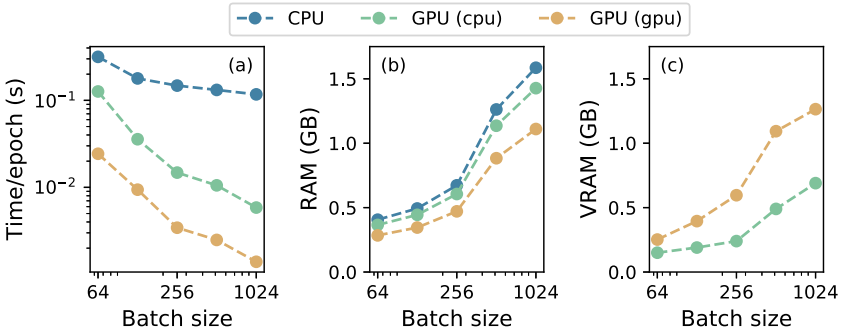


FIG. 2. Resources used for training the neural network on the energies of the TiO_2 database as a function of the batch size: (a) time per training epoch, (b) CPU RAM memory, and (c) GPU VRAM memory.

with force information. Regarding the former, the α parameter is bounded from 0 to 1, i.e., from pure energy training to only training on forces. As for the latter, some works in the field already suggest that including the forces of all the atoms of every structure in the reference set is not necessary to reach accurate predictions: Singraber *et al.* used from 0.41% to 4.1%,[48] and Artrith *et al.* stated that 10% was enough.[60,61] The results of our tests are depicted in Fig. 3, where both the energy and force RMSEs are displayed as a function of the fraction of structures whose forces have been used in the training. These structures are randomly selected from the whole training set.

First, the α parameter determines the balance between the accuracy of energies and forces. Lower values result in lower error of energies but higher error of forces, while higher values result in the opposite. For MD simulations, getting accurate forces is more important to ensure the stability of the simulations, whereas for Monte Carlo calculations, the energy is the main issue. Therefore, we need to find a balance between accuracy in energies and forces depending on the task at hand. It is important to note that the introduction of forces helps to generalize the prediction of energies in regions that are not included in the training examples,[38,43] so in most cases, the introduction of forces will be beneficial to some extent.

In our current example, the prediction error for forces decreases with increasing alpha, but for values above 0.1, the performance improves very little. In the case of the energies, all the models with α below 0.3 predict energies with an accuracy on the order of 1 meV/atom. Thus, we can conclude that $\alpha \in (0.1, 0.3)$ is the optimal range of values for a general-purpose potential for this system.

As shown in Fig. 3(b), the error of the forces does indeed decrease when adding more force information to the training set. However, this improvement diminishes when the percentage of forces included is high enough. This means that including from 10% to 20% is sufficient to achieve results close to the best performance that the model can provide.

Moreover, the number of forces included in the training examples heavily influences the computational requirements. This is shown in Fig. 4, where the training time, CPU RAM memory, and GPU VRAM memory for all three training strategies are displayed, this time as a function of the percentage of forces included. First, we note that including forces, even the smallest amount, increases at least 1 order of magnitude the time needed for training, which increases with the fraction of forces included. However, it is the memory that suffers the most with the addition of forces, rapidly increasing with it. This is somewhat mitigated by storing all the batch information in the CPU and only moving to the GPU the information one at a time. In any case, as Sec. IV will show, in most applications, including more than a small subset of the forces in the training set is not worth the computational cost.

It must be noted that in these calculations, all of the structures in the dataset are used for fitting the energy, whereas only a fraction of them are considered for training forces. Thus, let us investigate the importance of including the energy information of the structures that are not considered for force training. To do so, we have trained MLPs with only small portions of the structures of the dataset but training with all their forces in each case, i.e., excluding the energy-only structures from the training. Our results in Fig. 5 show that the removal of the remaining data for energy training heavily impacts

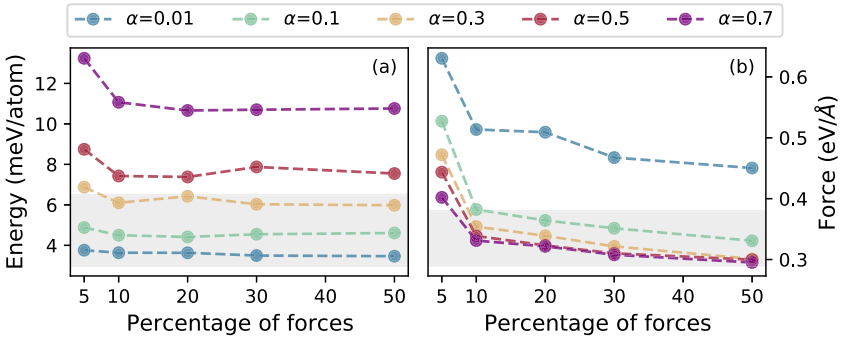


FIG. 3. Validation error of (a) energy and (b) forces as a function of the percentage of force information included in the training for different values of the weight parameter α for the TiO_2 database. The gray region is meant to guide the eye toward the set of parameters that we consider optimal.

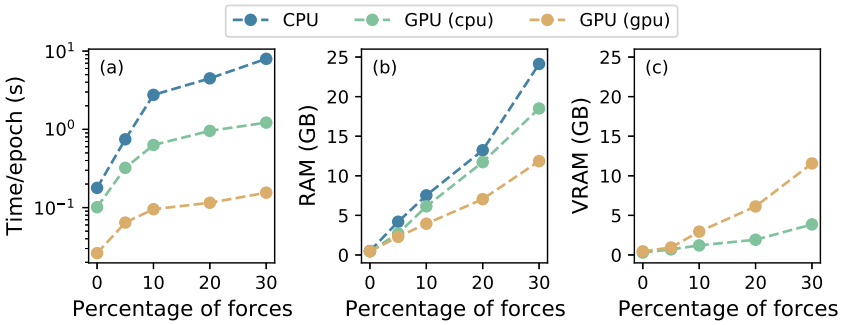


FIG. 4. Resources used for training the neural network on both energies and forces of the TiO_2 database as a function of the fraction of force information included: (a) time per training epoch, (b) CPU RAM memory, and (c) GPU VRAM memory.

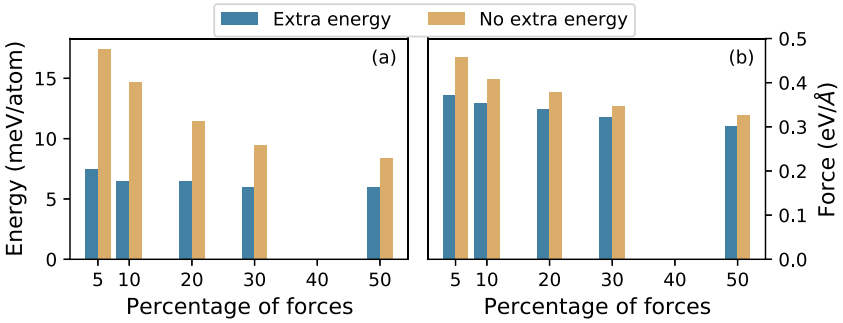


FIG. 5. (a) Energy and (b) force RMSE of the TiO_2 dataset for the two dataset convergence approaches. The blue bars correspond to the approach discussed so far, i.e., using the energies of all structures in the dataset for training and an increasing fraction of the atomic forces. The yellow bars indicate the results for an alternative approach, training on an increasing fraction of the structures but using all of the force and energy information.

the prediction of energies, especially when the percentage of data used is small. The prediction of forces is also affected, but not nearly as much. Thus, we conclude that adding force information helps generalize the energy prediction around the data points included in the training set, but for structures far from the reference ones, the prediction is not as accurate. Therefore, structures from all over the potential energy surface should be included in the training set to achieve accurate models.

IV. BENCHMARK ON OPEN DATA SETS

In this section, we will benchmark our code using several open databases and compare them with the results obtained by their authors. Unless otherwise stated, the descriptor used has been the Chebyshev polynomials with the same parameters as in the reference article, and so is the chosen architecture. Here, we will use $\alpha = 0.1$ to weigh the error for energy and force. Based on the analysis performed in Sec. III B, the prediction of energies is expected to be slightly less accurate than the one that would be achieved by fitting energies alone, but that will also result in more accurate force predictions.

A. Titanium dioxide

First, we summarize the results that have already been shown during this paper for the database of 7815 structures of several titanium dioxide phases developed by Artrith and Urban and which was used to test the original ænet code in its first release.[41] Reference energy and forces were obtained from DFT calculations using the Perdew–Burke–Ernzerhof (PBE) exchange–correlation functional.[62] In that work, the model was fitted only to the reference energies, and the descriptor used was the Behler–Parrinello symmetry functions.[51] The best fit resulted in an error of around 4 meV/atom in the energies, while the error on forces was not quantified in that work.

Our results are shown in Fig. 6, where we include the errors in energies and forces for both the training and testing sets. As we have already mentioned before, the error in energies slightly increases with the number of forces, while the one for forces drastically diminishes. The red dotted line shows the error obtained in the original work, which is slightly better than the one we get here for the energy. This is due to the choice of the α parameter; a similar accuracy to that of the original article could be obtained by reducing the value of that parameter, but this would come at the cost of increasing the error of the force prediction (for example, with $\alpha = 0.01$ in Fig. 4).

On the other hand, the error of the predicted absolute value of force converges to around 0.3 eV/Å with only 20% of forces included in the training. This error corresponds to around 2% of the mean force of the structures on the dataset (~20 eV/Å). Figure 6(c) shows the error in the direction of the predicted forces for different percentages. The addition of forces greatly improves its prediction, which is of great importance for MD simulations to be stable and accurate. Note that in cases where the absolute value of the force is smaller, the error in direction is more likely to be larger.[63]

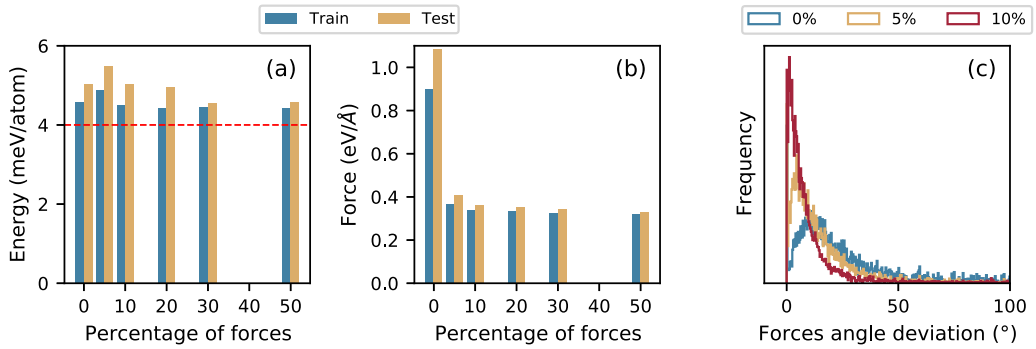


FIG. 6. (a) Energy and (b) force RMSE of the TiO_2 dataset for varying amount of force information. Blue for the training set and yellow for the validation set. The red dotted lines show the results obtained by the original authors of the dataset. (c) Distribution of the error of the direction of the predicted forces, computed as the angle between the reference and the predicted forces.

B. Liquid water

Second, we consider a database composed of 9189 liquid water structures, each with 192 atoms, whose energies and forces were evaluated with the revised PBE functional and with the addition of the Grimme D3 van-der-Waals correction.[64] This dataset was used by Chen *et al.* to test the performance of the LAMMPS and TINKER interfaces of ænet.[6] Water systems have been widely used as a benchmarking system for new developments in the field of MLPs[65–68] due to their complexity, so many open databases can be found. This one here is more of a challenge than the one for TiO_2, since the number of atoms in each structure is much higher and, therefore, so is the number of reference forces to be fitted. In this case, the authors also fitted the model to the reference energies.

This time, our results for energy training [Fig. 7(a)] are in excellent agreement with the fitting of Chen *et al.*, about 1 meV/atom. The error in forces is again lower in our fit, reducing about 50% with the smallest fraction of forces included. The absolute error of the forces cannot be directly compared to the results for the previous dataset, but the relative error is around 1% of the mean forces of the set, which is much lower in this case (around 2 eV/Å) than that of TiO_2. The most remarkable improvement comes from the angle deviation of forces, which becomes narrow around 0°.

C. Li–Mo–Ni–Ti oxide

Our next example is a much more complex quaternary oxide, LMNTO, a database consisting of 2616 bulk structures, with 56 atoms each. Here, the SCAN semilocal functional[69] was used to evaluate energies and forces. This reference dataset was used by Cooper *et al.* to test their method for including force information via Taylor series expansions,[63] so we have the reference value of the errors for fitting to forces in addition to energies in this case. Moreover, being a system composed of five elements, it is a great example of the case in which the Chebyshev polynomials excel as atomic fingerprints, since the resulting descriptor size does not depend on the total number of elements.[52]

Figure 8 demonstrates that our models perform equally well in predicting energies compared to the reference models but outperform them in terms of force prediction. With no force information, our models have a higher error than that of Cooper *et al.*; however,

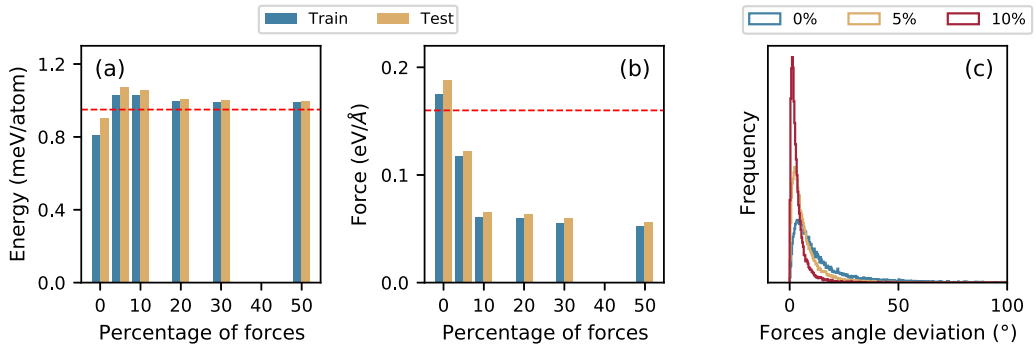


FIG. 7. (a) Energy and (b) force RMSE of the H_2O dataset as a function of the percentage of forces included in the training stage. Blue for the training set and yellow for the validation set. The red dotted lines show the results obtained by the original authors of the dataset. (c) Distribution of the error of the direction of the predicted forces.

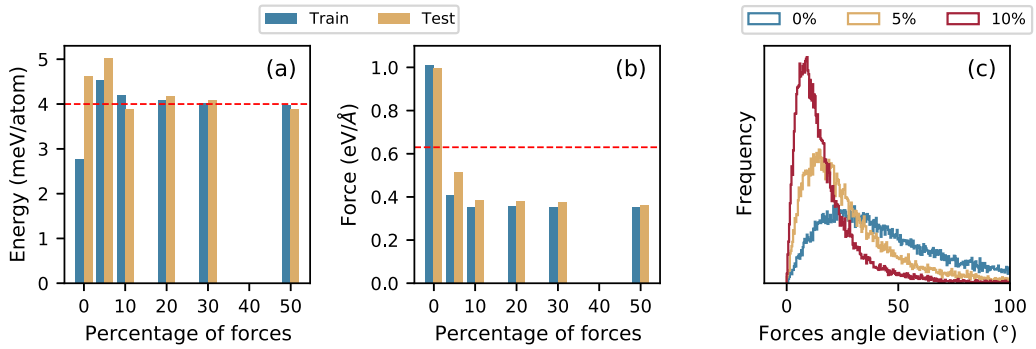


FIG. 8. (a) Energy and (b) force RMSE of the LMNTO dataset as a function of the percentage of forces included in the training stage. Blue for the training set and yellow for the validation set. The red dotted lines show the results obtained by the original authors of the dataset. (c) Distribution of the error of the direction of the predicted forces.

direct training on forces leads to improved performance over the Taylor series expansion. As in the two previous examples, the inclusion of forces initially increases energy error, but when sufficient force information is included, our models perform better than with only energy training. This highlights the benefit of incorporating forces in enhancing the generalization and transferability of the potentials.

D. Amorphous Li_xSi materials

The last database that we will consider consists of about 45 000 structures of amorphous Li_xSi, developed using a combination of density functional theory calculations, using the PBE functional, and evolutionary algorithms.[10] This set includes many phases with different stoichiometry, both bulk surfaces and nanoparticles. In this case, only energies are included in the database, so force training is not possible, but it will still be useful to see the performance of our code in a large database with many atoms per structure.

The best fit using ænet-PyTorch yields an error of 6.5 meV/atom in the training set and 7.6 meV/atom in the testing set, which is as good as the reference fit (6.3 meV/atom in training and 7.7 eV/atom in testing). As for the resources, 10 GB of memory was needed to fit the energies of the whole set with 128 structures per batch.

V. CONCLUSIONS

The last decade has shown that MLPs will play an important role in the study of new and complex materials at the atomic scale, and this has created a huge demand for tools to efficiently train such potentials. Training on GPUs is the standard practice in most fields of machine learning, and here, we presented an upgrade of the original ænet software to provide this capability. The ænet-PyTorch extension ensures that training the neural networks is no longer a bottleneck in the development of MLPs, even when accurate forces are required.

We demonstrated with different materials examples that ænet-PyTorch is efficient, particularly when training only on energies. The CPU version of the code is as fast as the original ænet implementation, and the GPU implementation reduces the training time by one to two orders of magnitude. Due to its compatibility with the ænet package, we expect this extension to have great synergy with the other features available in ænet, such as including force information in the training via a Taylor series expansion.[63]

If, on the other hand, direct training using atomic forces is desired, this is now feasible with the ænet-PyTorch code. We demonstrated that directly including force information in the training process is possible with ænet-PyTorch, thanks to the computationally efficient GPU implementation. Nonetheless, we strongly recommend users to include only a small fraction of forces in the training, since our benchmarks on systems with different numbers of atomic species, numbers of atoms, and dataset sizes demonstrated that accurate models can be obtained by including between 10% and 20% of the force information.

ACKNOWLEDGMENTS

This work was supported by the "Departamento de Educación, Política Lingüística y Cultura del Gobierno Vasco" (IT1458-22), the "Ministerio de Ciencia e Innovación" (Grant No. PID2019-106644GB-I00), and the Project HPC-EUROPA3 (Grant No. INFRAIA-2016-1-730897), with the support of the EC Research Innovation Action under the H2020 Programme. The authors acknowledge technical and human support provided by SGIker (UPV/EHU/ERDF, EU) and the Duch National e-Infrastructure and the SURF Cooperative for computational resources (National Supercomputer Snellius). J.L.-Z. acknowledges financial support from the Basque Country Government (PRE_2019_1_0025). N.A. acknowledges funding from the Bayer AG Life Science Collaboration ("!AIQU").

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Jon López-Zorrilla: Conceptualization (equal); Data curation (equal); Formal analysis (equal); Funding acquisition (equal); Investigation (equal); Methodology (equal); Software (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Xabier M. Aretxabaleta**: Data curation (equal); Formal analysis (equal); Investigation (equal); Writing – review & editing (equal). **Inwon Yeu**: Validation (equal); Writing – review & editing (equal). **Iñigo Etxebarria**: Conceptualization (equal); Supervision (equal); Writing – review & editing (equal). **Hegoi Manzano**: Conceptualization (equal); Funding acquisition (equal); Supervision (equal); Writing – review & editing (equal). **Nongnuch Artrith**: Conceptualization (equal); Funding acquisition (equal); Methodology (equal); Resources (equal); Software (equal); Supervision (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal).

DATA AVAILABILITY

ænet-PyTorch is open source and free for everyone to use and customize, as is the ænet package. The ænet-PyTorch code can be obtained from https://github.com/atomisticnet/aenet-PyTorch. Being written purely in Python and PyTorch, we believe that this code can be easily used for prototyping new techniques based on PyTorch features, such as custom loss functions, learning rate schedulers, and dropout layers to reduce overfitting.

REFERENCES

[1] J. Behler and M. Parrinello, "Generalized neural-network representation of high-dimensional potential-energy surfaces," Phys. Rev. Lett. **98**, 146401 (2007).
[2] K. Burke, "Perspective on density functional theory," J. Chem. Phys. **136**, 150901 (2012).
[3] L. Fiedler, K. Shah, M. Bussmann, and A. Cangi, "Deep dive into machine learning density functional theory for materials science and chemistry," Phys. Rev. Mater. **6**, 040301 (2022).
[4] R. F. Bishop and H. G. Kümmel, "The coupled-cluster method," Phys. Today **40**(3), 52 (1987).
[5] J. S. Smith, B. T. Nebgen, R. Zubatyuk, N. Lubbers, C. Devereux, K. Barros, S. Tretiak, O. Isayev, and A. E. Roitberg, "Approaching coupled cluster accuracy with a general-purpose neural network potential through transfer learning," Nat. Commun. **10**, 2903 (2019).
[6] M. S. Chen, T. Morawietz, H. Mori, T. E. Markland, and N. Artrith, "AENET–LAMMPS and AENET–TINKER: Interfaces for accurate and efficient molecular dynamics simulations with machine learning potentials," J. Chem. Phys. **155**, 074801 (2021).
[7] H.-A. Chen, P.-H. Tang, G.-J. Chen, C.-C. Chang, and C.-W. Pao, "Microstructure maps of complex perovskite materials from extensive Monte Carlo sampling using machine learning enabled energy model," J. Phys. Chem. Lett. **12**, 3591–3599 (2021).
[8] Y. Nagai, M. Okumura, K. Kobayashi, and M. Shiga, "Self-learning hybrid Monte Carlo: A first-principles approach," Phys. Rev. B **102**, 041124(R) (2020).
[9] A. Tirelli, G. Tenti, K. Nakano, and S. Sorella, "High-pressure hydrogen by machine learning and quantum Monte Carlo," Phys. Rev. B **106**, L041105 (2022).
[10] N. Artrith, A. Urban, and G. Ceder, "Constructing first-principles phase diagrams of amorphous Li_xSi using machine-learning-assisted sampling with an evolutionary algorithm," J. Chem. Phys. **148**, 241711 (2018).
[11] A. P. Bartók, J. Kermode, N. Bernstein, and G. Csányi, "Machine learning a general-purpose interatomic potential for silicon," Phys. Rev. X **8**, 041048 (2018).
[12] P. Rowe, G. Csányi, D. Alfe, and A. Michaelides, "Development of a machine learning potential for graphene," Phys. Rev. B **97**, 054303 (2018).
[13] P. Y. Chew and A. Reinhardt, "Phase diagrams—Why they matter and how to predict them," J. Chem. Phys. **158**, 030902 (2022).
[14] I. A. Kruglov, A. Yanilkin, A. R. Oganov, and P. Korotaev, "Phase diagram of uranium from *ab initio* calculations and machine learning," Phys. Rev. B **100**, 174104 (2019).
[15] K. Gubaev, E. V. Podryabinkin, and A. V. Shapeev, "Machine learning of molecular properties: Locality and active learning," J. Chem. Phys. **148**, 241727 (2018).
[16] G. Pilania, C. Wang, X. Jiang, S. Rajasekaran, and R. Ramprasad, "Accelerating materials property predictions using machine learning," Sci. Rep. **3**, 2810 (2013).
[17] E. V. Podryabinkin, E. V. Tikhonov, A. V. Shapeev, and A. R. Oganov, "Accelerating crystal structure prediction by machine-learning interatomic potentials with active learning," Phys. Rev. B **99**, 064114 (2019).
[18] J. Schmidt, J. Shi, P. Borlido, L. Chen, S. Botti, and M. A. L. Marques, "Predicting the thermodynamic stability of solids combining density functional theory and machine learning," Chem. Mater. **29**, 5090–5103 (2017).
[19] J. Behler, "Neural network potential-energy surfaces in chemistry: A tool for large-scale simulations," Phys. Chem. Chem. Phys. **13**, 17930–17955 (2011).
[20] J. Behler, "First principles neural network potentials for reactive simulations of large molecular and condensed systems," Angew. Chem., Int. Ed. **56**, 12828–12840 (2017).
[21] J. Behler, "Representing potential energy surfaces by high-dimensional neural network potentials," J. Phys.: Condens. Matter **26**, 183001 (2014).
[22] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, "Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons," Phys. Rev. Lett. **104**, 136403 (2010).
[23] W. J. Szlachta, A. P. Bartók, and G. Csányi, "Accuracy and transferability of Gaussian approximation potential models for tungsten," Phys. Rev. B **90**, 104108 (2014).
[24] A. P. Bartók, S. De, C. Poelking, N. Bernstein, J. R. Kermode, G. Csányi, and M. Ceriotti, "Machine learning unifies the modeling of materials and molecules," Sci. Adv. **3**, e1701816 (2017).
[25] S. T. John and G. Csányi, "Many-body coarse-grained interactions using Gaussian approximation potentials," J. Phys. Chem. B **121**, 10934–10949 (2017).
[26] V. Botu, R. Batra, J. Chapman, and R. Ramprasad, "Machine learning force fields: Construction, validation, and outlook," J. Phys. Chem. C **121**, 511–522 (2017).
[27] V. Botu and R. Ramprasad, "Learning scheme to predict atomic forces and accelerate materials simulations," Phys. Rev. B **92**, 094306 (2015).
[28] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning* (PMLR, 2017), pp. 1263–1272.
[29] L.-Y. Xue, F. Guo, Y.-S. Wen, S.-Q. Feng, X.-N. Huang, L. Guo, H.-S. Li, S.-X. Cui, G.-Q. Zhang, and Q.-L. Wang, "ReaxFF-MPNN machine learning potential: A combination of reactive force field and message passing neural networks," Phys. Chem. Chem. Phys. **23**, 19457–19464 (2021).
[30] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," Nat. Commun. **8**, 13890 (2017).
[31] A. P. Thompson, L. P. Swiler, C. R. Trott, S. M. Foiles, and G. J. Tucker, "Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials," J. Comput. Phys. **285**, 316–330 (2015).
[32] M. A. Wood and A. P. Thompson, "Extending the accuracy of the snap interatomic potential form," J. Chem. Phys. **148**, 241721 (2018).
[33] V. Botu and R. Ramprasad, "Adaptive machine learning framework to accelerate *ab initio* molecular dynamics," Int. J. Quantum Chem. **115**, 1074–1083 (2015).
[34] T. L. Jacobsen, M. S. Jørgensen, and B. Hammer, "On-the-fly machine learning of atomic potential in density functional theory structure optimization," Phys. Rev. Lett. **120**, 026102 (2018).

[35]C. Wang, K. Aoyagi, P. Wisesa, and T. Mueller, "Lithium ion conduction in cathode coating materials from on-the-fly machine learning," Chem. Mater. **32**, 3741–3752 (2020).
[36]R. Jinnouchi, F. Karsai, and G. Kresse, "On-the-fly machine learning force field generation: Application to melting points," Phys. Rev. B **100**, 014105 (2019).
[37]O. T. Unke, S. Chmiela, H. E. Sauceda, M. Gastegger, I. Poltavsky, K. T. Schütt, A. Tkatchenko, and K.-R. Müller, "Machine learning force fields," Chem. Rev. **121**, 10142–10186 (2021).
[38]X. Gao, F. Ramezanghorbani, O. Isayev, J. S. Smith, and A. E. Roitberg, "TorchANI: A free and open source PyTorch-based deep learning implementation of the ANI neural network potentials," J. Chem. Inf. Model. **60**, 3408–3415 (2020).
[39]A. Khorshidi and A. A. Peterson, "*Amp*: A modular approach to machine learning in atomistic simulations," Comput. Phys. Commun. **207**, 310–324 (2016).
[40]H. Wang, L. Zhang, J. Han, and W. E, "DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics," Comput. Phys. Commun. **228**, 178–184 (2018).
[41]N. Artrith and A. Urban, "An implementation of artificial neural-network potentials for atomistic materials simulations: Performance for TiO_2," Comput. Mater. Sci. **114**, 135–150 (2016).
[42]A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (2019).
[43]G. Imbalzano, A. Anelli, D. Giofré, S. Klees, J. Behler, and M. Ceriotti, "Automatic selection of atomic fingerprints and reference configurations for machine-learning potentials," J. Chem. Phys. **148**, 241730 (2018).
[44]F. Musil, A. Grisafi, A. P. Bartók, C. Ortner, G. Csányi, and M. Ceriotti, "Physics-inspired structural representations for molecules and materials," Chem. Rev. **121**, 9759–9815 (2021).
[45]M. Yaghoobi and M. Alaei, "Machine learning for compositional disorder: A comparison between different descriptors and machine learning frameworks," Comput. Mater. Sci. **207**, 111284 (2022).
[46]A. P. Bartók, R. Kondor, and G. Csányi, "On representing chemical environments," Phys. Rev. B **87**, 184115 (2013).
[47]S. De, A. P. Bartók, G. Csányi, and M. Ceriotti, "Comparing molecules and solids across structural and alchemical space," Phys. Chem. Chem. Phys. **18**, 13754–13769 (2016).
[48]A. Singraber, T. Morawietz, J. Behler, and C. Dellago, "Parallel multistream training of high-dimensional neural network potentials," J. Chem. Theory Comput. **15**, 3075–3092 (2019).
[49]A. M. Miksch, T. Morawietz, J. Kästner, A. Urban, and N. Artrith, "Strategies for the construction of machine-learning potentials for accurate and efficient atomic-scale simulations," Mach. Learn.: Sci. Technol. **2**, 031001 (2021).
[50]A. Krogh and J. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems 4* (1991).
[51]J. Behler, "Atom-centered symmetry functions for constructing high-dimensional neural network potentials," J. Chem. Phys. **134**, 074106 (2011).
[52]N. Artrith, A. Urban, and G. Ceder, "Efficient and accurate machine-learning interpolation of atomic energies in compositions with many species," Phys. Rev. B **96**, 014112 (2017).
[53]A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in' t Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS—A flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," Comput. Phys. Commun. **271**, 108171 (2022).
[54]J. A. Rackers, Z. Wang, C. Lu, M. L. Laury, L. Lagardère, M. J. Schnieders, J.-P. Piquemal, P. Ren, and J. W. Ponder, "Tinker 8: Software tools for molecular design," J. Chem. Theory Comput. **14**, 5273–5289 (2018).
[55]H. Kimizuka, B. Thomsen, and M. Shiga, "Artificial neural network-based path integral simulations of hydrogen isotope diffusion in palladium," J. Phys.: Energy **4**, 034004 (2022).
[56]A. Hjorth Larsen, J. Jørgen Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus *et al.*, "The atomic simulation environment—A Python library for working with atoms," J. Phys.: Condens. Matter **29**, 273002 (2017).
[57]D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980 (2014).
[58]M. D. Zeiler, "ADADELTA: An adaptive learning rate method," arXiv:1212.5701 (2012).
[59]I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," arXiv:1711.05101 (2017).
[60]N. Artrith and J. Behler, "High-dimensional neural network potentials for metal surfaces: A prototype study for copper," Phys. Rev. B **85**, 045439 (2012).
[61]N. Artrith, B. Hiller, and J. Behler, "Neural network potentials for metals and oxides—First applications to copper clusters at zinc oxide," Phys. Status Solidi B **250**, 1191–1203 (2013).
[62]J. P. Perdew, K. Burke, and M. Ernzerhof, "Generalized gradient approximation made simple," Phys. Rev. Lett. **77**, 3865 (1996).
[63]A. M. Cooper, J. Kästner, A. Urban, and N. Artrith, "Efficient training of ANN potentials by including atomic forces via Taylor expansion and application to water and a transition-metal oxide," npj Comput. Mater. **6**, 54 (2020).
[64]S. Grimme, J. Antony, S. Ehrlich, and H. Krieg, "A consistent and accurate *ab initio* parametrization of density functional dispersion correction (DFT-D) for the 94 elements H-Pu," J. Chem. Phys. **132**, 154104 (2010).
[65]A. P. Bartók, M. J. Gillan, F. R. Manby, and G. Csányi, "Machine-learning approach for one- and two-body corrections to density functional theory: Applications to molecular and condensed water," Phys. Rev. B **88**, 054104 (2013).
[66]B. Cheng, E. A. Engel, J. Behler, C. Dellago, and M. Ceriotti, "Ab initio thermodynamics of liquid and solid water," Proc. Natl. Acad. Sci. U. S. A. **116**, 1110–1115 (2019).
[67]V. Quaranta, J. Behler, and M. Hellström, "Structure and dynamics of the liquid–water/zinc-oxide interface from machine learning potential simulations," J. Phys. Chem. C **123**, 1293–1304 (2018).
[68]A. Singraber, J. Behler, and C. Dellago, "Library-based LAMMPS implementation of high-dimensional neural network potentials," J. Chem. Theory Comput. **15**, 1827–1840 (2019).
[69]J. Sun, A. Ruzsinszky, and J. P. Perdew, "Strongly constrained and appropriately normed semilocal density functional," Phys. Rev. Lett. **115**, 036402 (2015).