I. INTRODUCTION

Inspired by the brain of living creatures\(^1\) and empowered through their universal function approximation capabilities\(^2\), artificial neural networks (ANNs) provide an unbiased approach to a large variety of applications, were the exact mapping between input and output remains unknown or computationally unfeasible. Due to their flexible topology, optimized w.r.t. a particular task, ANNs have already worked their way into a wide variety of fields ranging from image processing\(^3\) and speech recognition\(^4–6\) to the construction of high-dimensional potential-energy surfaces\(^7\). Neural networks have successfully expanded the field of artificial intelligence, while traditionally being prosperous in addressing intellectually demanding problems for humans defined by well-known mathematical rules, to the domain of intuitive problems whose translation into a mathematical framework of instructions remains challenging\(^8\).

Since its proposal by Walter Kohn, density functional theory (DFT)\(^9\) went through an incredible success story, establishing itself as part of the standard repertoire for electronic structure calculations in the fields of chemistry, physics and materials science. However, its computationally demanding nature renders the method ill-suited for extended systems or long timescales of molecular dynamics (MD). Approximate descendants of DFT, such as density functional tight binding (DFTB)\(^10\), strive to fill the gap between \textit{ab initio} DFT and fully empirical forcefields, but come at the expense of introducing parameters whose values need to be determined, usually by fitting to \textit{ab initio} theory or experiment.

Starting more than two decades ago, the first machine learning feed-forward neural network potentials were constructed to infer global properties of systems with a fixed number of degrees of freedom\(^11\). The major limitations of these first-generation neural network potentials included the limited number of dimensions and non-compliance with physical conservation laws regarding permutation symmetry w.r.t. atoms, as well as translational and rotational energy conservation. Second-generation potentials\(^7\), which build upon the ideas of J. Behler and M. Parrinello, define a topology of atomic neural networks to construct high-dimensional potential-energy surfaces that overcome the limitations of the first generation. Due to their near-sighted nature and focus on short-range interactions, efforts have been made to extend the second-generation potentials by introducing additional sets of atomic neural networks, considering long-ranged contributions like electrostatic energies as well as non-locality\(^12–14\).

\textit{Fortnet}\(^15\) is a Behler-Parrinello neural network implementation, written in Fortran 2008. Its core features include parallelized training of fully connected feed-forward neural networks that are combined to form a Behler-Parrinello topology, the fundamental ideas of which are outlined in section II B. Data driven MPI-parallelism combined with HDF5\(^16\) based I/O enables to surpass the limits of shared-memory architectures and renders Fort-
net well-suited for large datasets and network sizes. Although Fortran still plays a rather minor role in the machine learning community, overshadowed by the popularity of frameworks like Tensorflow, Keras, Py-Torch or Scikit-learn, there are appealing characteristics inherent to a statically and strongly typed language with efficient whole-array arithmetic that support this fundamental design decision. This was also recently pointed out, and demonstrated, in the so-called neural-fortran implementation of Curcic. Since (formatted) I/O is arguably a major weakness of Fortran, a Python framework was developed to handle the generation of the HDF5 datasets as well as the extraction of all desired results out of the likewise HDF5 based output of Fortnet.

With Ammothum Kandy et al. implementing the curvature constrained splines (CCS) methodology to provide a framework that enables fitting DFTB repulsive potentials with minimal human effort, another robust tool for the construction of two-body parametrizations became available. However, the tool further exposes the inability of a two-body description to capture the subtle changes in energetics when transferring between significantly different chemical environments, as obtained for silicon phases of varying coordination. To overcome these limitations, Ammothum Kandy et al. managed to achieve full energetic transferability by combining self-consistent DFTB (SCC-DFTB) with a near-sighted ANN, hereafter referred to as the DFTB+ANN approach. Focus now shifts to identifying which energy terms can be trained efficiently and with sufficient accuracy. In particular, the question arises, whether the provision of an explicitly calculated DFTB energy term can improve the accuracy and the transferability of an ANN, and whether the increased computational time due to the diagonalization of the associated DFTB Hamiltonian can be justified. We have investigated this question using a dataset consisting of various periodic silicon structures, which contains, among others, challenging point defects, for which standard DFTB parametrizations fail to achieve a satisfactory agreement with PBE-DFT.

II. THEORY

A. Feed-forward neural networks

An artificial neural network (ANN) is a (directed) graph $G = (V, E)$, whose vertices $v \in V$ are referred to as units or neurons, connected by edges $e \in E$. The total set $V$ of vertices is divided into input $V_{\text{in}}$, hidden $V_{\text{hidden}}$ and output subset $V_{\text{out}}$, that is, $V = V_{\text{in}} \cup V_{\text{hidden}} \cup V_{\text{out}}$ as well as $V_{\text{in}}, V_{\text{out}} \neq \emptyset$ and $V_{\text{hidden}} \cap (V_{\text{in}} \cup V_{\text{out}}) = \emptyset$. Each connection $(v, u) \in E$ is assigned a weight $\omega_{uv}$. Feed-forward neural networks (FFNNs) constitute an acyclic graph, i.e. backward connections are absent.

The multilayer perceptron is often referred to as an ordinary feed-forward neural network: Consider a graph $G = (V, E)$ satisfying $V_{\text{in}} \cap V_{\text{out}} = \emptyset$, $V_{\text{hidden}} = V_{\text{hidden}}^1 \cup \cdots \cup V_{\text{hidden}}^{n-2}$ whereby $V_{\text{hidden}}^i \cap V_{\text{hidden}}^j = \emptyset$, $\forall 1 \leq i < j \leq n-2$. The set of edges is given by $E \subseteq (V_{\text{in}} \times V_{\text{hidden}}^1) \cup (\bigcup_{i=1}^{n-3} V_{\text{hidden}}^i \times V_{\text{hidden}}^{i+1}) \cup (V_{\text{hidden}}^{n-2} \times V_{\text{out}})$. In the case that the hidden neurons form a null set, that is $n = 2$, $V_{\text{hidden}} = \emptyset$, it holds that $E \subseteq V_{\text{in}} \times V_{\text{out}}$.

Therefore, the general $n$-layer perceptron maps a set of input values to a set of output values, with any additional processing taking place in $n - 2 \geq 0$ hidden layers. Neurons of a given layer are fully connected to the neurons of adjacent layers. Additionally, every node $v \in V_k$ of layer $1 < k \leq n$ is assigned an offset (bias) parameter $b^k_v$, which shifts the argument of the respective activation function $\sigma$. Figure 1 illustrates the general topology of such networks.

![Diagram of a general n-layer perceptron](image)

**FIG. 1.** General n-layer perceptron. The i input values $y^1$ are mapped to j output values $y^n$. Between the input and output layer, $n - 2 \geq 0$ hidden layers, whose neurons are fully connected to the neurons of adjacent layers, further process the input by determining their weighted sum and applying a suitable activation function. Any node of layer $1 < k \leq n$ is assigned an additional offset (bias) $b^k_v$, shifting the argument $z^k$ of the activation function.

The actual evaluation is carried out by a forward propagation of the input through the individual layers of the network. The network input $z^k$ of each neuron of layer $1 < k \leq n$, is a weighted sum of the previous raw outputs $y^{k-1}$:

$$z^k = (\mathbf{w}^k) ^\top y^{k-1} + b^k$$

(1)

with weight coefficients $\mathbf{w}^k$, biases $b^k$, the output

$$y^k = \sigma^k(z^k)$$

(2)

and the (usually non-linear) activation functions $\sigma^k$.

The quality of the predictions is usually measured by a scalar cost function $C$. The gradients of the cost function in weight-bias space can be obtained via the back-propagation algorithm. Using the error $\delta^k_i$ of neuron $i$ in layer $k$

$$\delta^k_i \equiv \frac{\partial C}{\partial z^k_i}$$

(3)
the derivatives can be written as\(^{26}\)
\[
\delta^k = \nabla_{y_k} C \circ \left(\sigma^k(z^k)\right)^{'}
\]
\[
= \left((w^{k+1})^\top \delta^{k+1}\right) \circ \left(\sigma^k(z^k)\right)^{'}
\]
\[
\frac{\partial C}{\partial w^k_{ij}} = \frac{\partial C}{\partial z^k_i} \frac{\partial z^k_i}{\partial w^k_{ij}} = \delta^k y_j^{-1}
\]
\[
\frac{\partial C}{\partial b^k_i} = \frac{\partial C}{\partial z^k_i} \frac{\partial z^k_i}{\partial b^k_i} = \delta^k_i
\]
where \(\circ\) denotes the element-wise multiplication and prime the derivative with respect of the function argument.

The training cycle usually contains the following steps\(^{26}\):

1. **Input Features:** Set the neuron activation vector \(y^1\) of the input layer, provided by the current dataset entry.

2. **Forward-Propagation:** Propagate the activation vector \(y^1\) through the subsequent layers \(2 \leq k \leq n\), by calculating (and caching) \(z^k\) as well as \(y^k\).

3. **Output Error:** Based on the forward-pass, determine \(\delta^n = \nabla_{y^n} C \circ \left(\sigma^n(z^n)\right)^{'}\).

4. **Backward-Propagation:** Propagate the error \(\delta^k\) through the network, starting with the adjacent layer of the output layer, that is, \(k = n-1, n-2, \ldots, 2\).

5. **Cost Gradients:** Calculate the gradients of the cost function in the weight-bias space
\[
\nabla \frac{\partial C}{\partial w^k_{ij}} = \delta^k y_j^{-1}, \nabla \frac{\partial C}{\partial b^k_i} = \delta^k_i.
\]

6. **Network Update:** Predict updated network parameters by using a suitable gradient-based optimizer.

**B. Behler-Parrinello neural networks**

The neural network topology implemented in Fortnet was proposed by J. Behler and M. Parrinello\(^7\). It strives to overcome limitations of ordinary feed-forward neural networks (FFNNs), i.e. the multilayer perceptron, trained on atomic coordinates. The final result, e.g. total energy \(E\), is decomposed into atomic contributions \(E = \Sigma = \sum_i E_i\) and assembled from a topology composed of multiple element related subnets \(S_j\). The Behler-Parrinello neural network (BPNN), as presented by Figure 2, is tightly bound to the atom-centered symmetry functions (ACSF)\(^{27}\), which map the raw coordinates of the atoms to values that reflect their local environment w.r.t. neighboring atoms. By definition, the ACSF ensure that translations or rotations of a system remain energy-conserving operations, regardless of the change in the absolute coordinates, just like permuting atoms of the same element.

The Behler-Parrinello approach leads to a somewhat involved training process. The initial weights of each subnet are usually chosen randomly, in conjunction with vanishing bias. In preparation for the actual training iterations, the ACSF values are calculated by considering all atoms of the dataset. Subsequently, the subnets are fed one atom’s ACSF set at a time, while temporarily caching the arguments of the activation functions. After feeding the network with all atoms of a certain structure, the output is obtained as a summation of atomic contributions (when fitting on system-wide properties). Once every datapoint is evaluated, the atomic gradients of a suitable cost function in weight-bias space are determined by back-propagation (cf. section II A) and condensed into a total gradient, based on which new parameters are predicted by a gradient-based optimizer.

**III. IMPLEMENTATION**

**A. Data parallelism**

Since Fortnet is designed to efficiently handle large datasets and network sizes, all performance critical routines are MPI-parallelized and work on both shared and distributed memory architectures. The most important parallel routines include the ACSF feature generation, network evaluation (forward-propagation) and gradient calculation (backward-propagation). Fortnet (v0.2) was benchmarked by using a dataset of 15000 single-element geometries, holding roughly 64 atoms per 3-dimensional periodic supercell. The sub-network architecture was chosen to be \([9-10-10-1]\), having 9 ACSF input nodes with a cutoff of \(R_c = 5\text{Å}\), two hidden layers with 10 neurons as well as a single output cell. This topology corresponds to a total number of 221 fitting parameters. Figure 3 illustrates the respective I/O-less CPU times...
over the number of MPI processes, performing $2 \cdot 10^4$ full-batch steepest-descent steps and initial ACSF generation.

![Efficiency](chart.png)

FIG. 3. Illustration of Fortnet’s MPI-parallelization. Both, the I/O-cleaned CPU time as well as parallel efficiency reduce for an increasing number of MPI processes. The computational details of the benchmark are outlined in section III A.

Until approximately 100 processes, the parallel efficiency is quite satisfactory. Going beyond this core count, the CPU time slowly saturates, i.e. the serial parts of the code take over a larger relative impact on the runtime, due to the strongly accelerated gradient calculation. This leads to a reduction of the computational load and behaves as expected, considering the batch size at hand. The serial parts mainly contain the network updates by the respective optimizer. In fact, the MPI-parallelization may be Fortnet’s greatest strength, since execution on distributed-memory machines is seamlessly possible for arbitrary core-counts.

## B. Program Features

In the following, some central features of the program package are shortly presented. Further details can be found in the user documentation.\(^{28}\)

### 1. Input Features

**a. Atom-centered Symmetry Functions** Atom-centered Symmetry Functions (ACSFs)\(^{27}\) are implemented to generate input features that represent local atomic environments, by incorporating geometric information. The original paper of J. Behler proposed five so-called $G$-functions, parametrized by using suitable sets of $(R_c, \eta, R_s, \kappa, \xi, \lambda)$ values. The function $f_c$ cuts off the sphere that regards neighboring atoms in a radius of $R_c^{27}$:

$$f_c(R_{ij}) = \begin{cases} 
\frac{1}{2} \left[ \cos \left( \frac{\pi R_{ij}}{R_c} \right) + 1 \right] & \text{for } R_{ij} \leq R_c \\
0 & \text{for } R_{ij} > R_c 
\end{cases}$$

An element-resolved scheme as presented above may result in a large number of input nodes $N_{in}$, originating from different functions for each element (two-body ACSFs) or element-pair (three-body ACSFs) respectively:

$$N_{in} = N_{\text{elements}} \sum_{i=1}^{5} N_{Gi} + \frac{N_{\text{elements}}!}{2!(N_{\text{elements}}-2)!} \sum_{i=4}^{5} N_{Gi} \quad (14)$$

Therefore Fortnet provides an element-unresolved scheme whose neighbor lists are element-agnostic (cf. supplementary material I, listing 1). Even considering the fact that the BPNN consists of multiple subnetworks, dedicated to the respective elements of the dataset, the raw ACSF are expected to represent contradictory data in this particular case. In order to overcome this contradiction, a similar strategy as weighted ACSF (wACSF)\(^{29}\) is pursued to recover discernment between different elements. A large variety of atomic properties such as the atomic number, number of valence electrons, Hubbard $U$’s, Mulliken populations or any other suitable quantity would be conceivable to improve atomic characterization. It may be ensured that all summands have an equal sign in order to prevent cancellation of contributions. While this is not necessarily problematic, it should be handled with caution\(^{27}\).
The parametrization of the G-functions may be carried out by an automatic ACSF parameter generation scheme that aims to cover the cutoff sphere as evenly as possible, with the number of symmetry functions available. This reduces the user input to the desired cutoff radius $R_c$ in conjunction with the number $N_{\text{rad}}$ of radial $G^2$ and $N_{\text{ang}}$ angular $G^5$ functions (cf. supplementary material I, listing 2). However, each individual function can also be defined manually and for example employed to specifically extend the automatic generation scheme (cf. supplementary material I, listing 3).

b. External Atomic Input Features Since this kind of software can never claim that all conceivable feature generators are implemented natively and to vastly improve the flexibility of Fortnet, there is no need to solely rely on ACSF to characterize the local environment of each atom. Rather, the independent specification of external atomic input features, extracted from the dataset, is possible (cf. supplementary material I, listing 4). In the context of electronic structure calculations, properties such as the atomic number, number of valence electrons, Hubbard $U$’s, Mulliken populations or any other suitable quantity may thus be included as input feature.

2. Network Initialization Scheme

The network initialization is carried out by a truncated Xavier initialization$^{30}$. This choice primarily refers to the type of sigmoidal activation functions used to generate the results of section IV. Its main idea is to draw the weight vector $w^k$ of layer $k$ from a normal distribution with zero mean and a variance $V$ that depends on the number of nodes $n^k$ of the (adjacent) layer $k$, while using vanishing bias values $b^k$. As shown by Glorot and Bengio$^{30}$, this particular choice of variance refers to the separating case between vanishing and exploding gradients and imposes variance preservation across all layers, i.e. $V(y^k) = V(y^{k-1})$. To obtain a suitable weight distribution, a luxury random number generator$^{31-33}$ draws high-quality pseudo-random numbers $x$ from an interval $x \in [x_-, x_+]$, defined by the hyperparameter $N_{\text{min}}$ of the truncation:

$$N(\mu, \sigma^2) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

$$\sigma^2 \equiv V(w^k) = \frac{2}{n^k-1 + n^k}$$

$$x_\pm = \pm \sqrt{ -2\sigma^2 \ln \left( N_{\text{min}} \sigma \sqrt{2\pi} \right) }$$

$$\implies \quad w^k_{ij} = g \cdot N(\mu = 0, \sigma^2), \quad b^k_i = 0$$

As for $N_{\text{min}}$, the scaling factor $g$ is currently not accessible from the user perspective, although the implementation of this would be trivial, and is always chosen to be one.

3. Forward- and Backward-Propagation

Similar to other machine learning codes, Fortnet heavily relies on its implementation of the forward- and backward-propagation, with the network evaluation as well as the calculation of the cost-gradients in weight-bias space naturally influencing the training process. The realization of eq. (2), eq. (4), (5), (6) and (7) in terms of Fortran’s array arithmetic forms the core of both algorithms.

The implementation was inspired by similar works$^{34,35}$, and the neural-fortran$^{21}$ paper. Latter demonstrates the competitiveness of such a bare-bones implementation, compared to the Keras$^{18}$ library with TensorFlow$^{17}$ as backend, especially highlighting the significantly reduced memory consumption. In contrast to the neural-fortran$^{21}$ implementation, Fortnet generalizes the back-propagation implementation to arbitrary cost-function gradients and exploits the fact that the analytical derivatives of the implemented cost functions are known. To successively optimize the weight and bias network parameters, based on the gradients obtained by back-propagation, Fortnet provides several minimization algorithms.

4. Activation Functions

To enable non-linear potentials, among others, logistic, arcsin and hyperbolic tangent, (leaky) ReLU$^{36}$ and softplus$^{37}$ transfer is available. While sigmoidal activation functions have historically had great relevance and are still frequently used for this purpose, they suffer from soft saturation and vanishing gradients, especially in inappropriately initialized$^{30}$ networks. In order to circumvent these problems, rectified linear units (ReLU)$^{36}$ became increasingly popular, in particular due to significant speedups$^{38}$ achieved while training deep (convolutional) neural networks. The modular structure of the implementation, using abstract interfaces and procedure pointers, allows for a convenient extension of this list.

5. Atomic and Global Training Targets

The implemented Behler-Parrinello topology can either be trained on atomic properties like forces and charges, or on system-wide properties such as the total energy of a geometry. A mixture of both types of targets will be subject of future developments.

6. Loss Functions and Regularization

In terms of the maximum likelihood framework$^{8}$, the minimization of the negative log-likelihood of a neural network with linear output units is carried out by minimizing the residual sum of squares, i.e. mean squared
error. Fortnet offers the mean squared loss as default loss function, with several alternatives:

- mean squared error (mse)
  \[ L = \frac{1}{N} \sum_{i=1}^{N} (y_{i}^{\text{ref}} - y_{i}^{\text{nn}})^2 \]
- root mean square error (rmse)
  \[ L = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_{i}^{\text{ref}} - y_{i}^{\text{nn}})^2} \]
- mean absolute error (mae)
  \[ L = \frac{1}{N} \sum_{i=1}^{N} |y_{i}^{\text{ref}} - y_{i}^{\text{nn}}| \]
- mean absolute percentage error (mape)
  \[ L = \frac{100}{N} \sum_{i=1}^{N} \left| \frac{y_{i}^{\text{ref}} - y_{i}^{\text{nn}}}{y_{i}^{\text{ref}}} \right| \]

Adding further loss functions is straightforward.

Regularization aims to improve upon the generalization error, while leaving the training error unaltered. Besides utilizing a plain early-stopping mechanism to avoid overfitting, loss-based regularization\textsuperscript{39–41} puts a soft constraint onto the weights of a network, while compromising between a reduced variance and increased bias. The original cost function \( C_{0} \) is penalized by an additional term \( \tilde{C} \), whereas its influence is adjusted by a positive hyperparameter \( \lambda \), that can be set from the user’s perspective:

\[ C = C_{0} + \lambda \tilde{C} \quad (15) \]

Fortnet provides three popular loss-based regularization techniques to induce weight decay:

- \( L^{1} \) - Lasso\textsuperscript{30} 
  \[ \tilde{C} = \frac{1}{n} \sum_{i=1}^{n} |w_{i}|, \quad \frac{\partial \tilde{C}}{\partial w_{i}} = \frac{\text{sgn}(w_{i})}{n} \]
- \( L^{2} \) - Ridge\textsuperscript{40} 
  \[ \tilde{C} = \frac{1}{2n} \sum_{i=1}^{n} w_{i}^{2}, \quad \frac{\partial \tilde{C}}{\partial w_{i}} = w_{i} / n \]
- Elastic Net\textsuperscript{41} 
  \[ \tilde{C} = \frac{1}{n} \left( \frac{1-\alpha}{2} \sum_{i=1}^{n} w_{i}^{2} + \alpha \cdot \sum_{i=1}^{n} |w_{i}| \right) \]
  \[ \frac{\partial \tilde{C}}{\partial w_{i}} = \frac{1}{n} \left( (1-\alpha) \cdot w_{i} + \alpha \cdot \text{sgn}(w_{i}) \right) \]

\( L^{1} \) (lasso) an \( L^{2} \) (ridge) regularization encourage continuously shrinking weight coefficients, while training the network. The qualitative difference of lasso regression w.r.t. ridge regression is the promotion of sparsity, i.e. automatic feature selection\textsuperscript{41}, whereas an \( L^{2} \) penalty is not capable of deactivating neurons. Notwithstanding this, there are cases in which the feature selection by lasso empirically proved unreliable\textsuperscript{41} and the prediction accuracy of ridge-penalized networks prevails. To combine the strengths of both approaches, the elastic net\textsuperscript{41} regularization got introduced as a mixture of lasso and ridge regression. The special cases \( \alpha = 0 \) and \( \alpha = 1 \) are equivalent to \( L^{2} \) and \( L^{1} \) regularization respectively.

7. Atomic Forces

Atomic forces, i.e. the negative gradients of the total energy \( E \) w.r.t. the coordinates \( R_{i}^{\alpha} \), \( \alpha = (x,y,z) \) of atom \( i \), may be determined numerically or analytically. Former case is based on the central finite difference:

\[ F_{i}^{\alpha} = -\frac{\partial E}{\partial R_{i}^{\alpha}} \approx \frac{E(R_{i}^{\alpha} + \Delta R_{i}^{\alpha}) - E(R_{i}^{\alpha} - \Delta R_{i}^{\alpha})}{2\Delta R_{i}^{\alpha}} \quad (16) \]

Although the associated computation of \( 6N \) ACSF sets per datapoint of \( N \) atoms is MPI-parallelized, an analytical computation based on the BPNN’s forward derivative\textsuperscript{42} is more advantageous. Since the ACSF are explicitly known and differentiable, the analytical force components emerge by applying the chain rule\textsuperscript{27}:

\[ F_{i}^{\alpha} = -\sum_{i=1}^{N} \frac{\partial E_{i}}{\partial R_{i}^{\alpha}} = -\sum_{i=1}^{N} \sum_{j=1}^{M_{i}} \frac{\partial E_{i}}{\partial G_{i,j}} \frac{\partial G_{i,j}}{\partial R_{i}^{\alpha}} \quad (17) \]

The second sum treats \( M_{i} \) ACSF functions that represent the local environment of the \( i \)-th atom. The first derivatives\textsuperscript{42} of the network output \( y^{k}(x) \in \mathbb{R}^{m_{k}} \) w.r.t. the input features \( x \in \mathbb{R}^{m_{1}} \) may be defined as Jacobian matrix \( J^{n}(x) \in \mathbb{R}^{(m_{n} \times m_{1})} \) of layer \( k = n \):

\[ J^{n}(x) = \nabla y^{n}(x) = \text{diag} \left[ \alpha^{m_{n}} \left( (w^{n})^{\top} y^{n-1}(x) + b^{n} \right) \right] \times \left[ (w^{n})^{\top} J^{n-1}(x) \right] \quad (18) \]

To obtain \( J^{n-1}(x) \), a recursion is carried out \( \forall k = 2, \ldots, n-1 \), originating from the identity \( J^{1}(x) = I \in \mathbb{R}^{m_{1} \times m_{1}} \). Explicit expressions for the ACSF derivatives can be found in appendix A.

We have benchmarked the analytical force calculation by using datasets of \( 10^{3}, 10^{4} \) and \( 10^{2} \) silicon carbide geometries, holding 64 atoms per 3-dimensional periodic supercell. The sub-network architectures were chosen to be \( [32-3-3-1] \), having 32 ACSF input nodes with a cutoff of \( R_{c} = 5 \text{Å} \), two hidden layers with 3 neurons as well as a single output node. This topology corresponds to a total number of 230 fitting parameters. Figure 4 illustrates the relevant CPU times over the number of dataset geometries, performing analytical and numerical force analysis.

The analytical and numerical force related, I/O-cleaned CPU time increases linearly with the number of dataset geometries. However, the total timings of the two methods differ significantly, showing the vast advantage of analytical expressions for the Jacobi matrix of the network and the ACSF derivatives w.r.t. the atomic coordinates, over numerical central finite differences.

8. Generating a Dataset

As already mentioned in section I, a Python framework handles the composition of compatible datasets and
enables the extraction of selected results after a Fortnet run has been completed. Thereby it relies on the strength of the Python package Atomic Simulation Environment (ASE) to represent geometric information that may have previously been extracted from the outputs of the employed simulation package. The actual dataset generation is as simple as illustrated by code listing 5 of the supplementary materials. By invoking the referenced code, an HDF5 dataset will be written to disk, that includes geometric information for later ACSF generation, external atomic input features as well as system-wide training targets.

9. Extracting Results

Analogous to the dataset generation, extracting results from the HDF5 output is carried out via a Python layer (cf. supplementary material I, listing 6). To fetch desired entries the class provides several properties that may be extracted, including but not limited to the mode of the run that produced the output file (validation or system-wide), the predictions of the network potential as well as corresponding targets if provided.

10. Interfacing with external codes

Fortnet can currently be interfaced with other software packages using foremost file communication. (Future developments will focus on establishing an API that enables library interfacing to reduce the I/O overhead.) The external driver creates necessary input files for Fortnet and starts an individual Fortnet program for each of the inputs. After Fortnet has finished, the driver analyses the created output files and extracts the necessary information from those. A file communication based interface to the Atomic Simulation Environment (ASE) package is available.

IV. DFTB REPULSIVE PARAMETRIZATION

In this section we demonstrate the capabilities of the Fortnet package by training and evaluating ANNs for a combined DFTB+ANN approach, where the force-field like contribution of the DFTB method is replaced or corrected by an ANN.

A. DFTB basics

The density functional tight binding (DFTB) fills the gap between ab initio DFT electronic structure methods and fully empirical force-fields in the domain of materials and chemical modeling. Foulkes and Haydock represented the ground-state density \( \rho(r) \) as a perturbed reference density \( \rho_0(r) \), i.e. emerging from a superposition of neutral atomic densities. The self-consistent charge (SCC) extension originates from a second-order expansion of the Kohn-Sham total energy w.r.t. atomic charge fluctuations \( \delta \rho \) around a reference density \( \rho_0 \):

\[
E[\rho_0 + \delta \rho] = E[\rho_0] + E^1[\rho_0, \delta \rho] + E^2[\rho_0, \delta \rho^2] \quad (19)
\]

\[
E_{\text{rep}}[\rho_0] = E_{\text{elec}}[\rho_0, \delta \rho, \delta \rho^2] + E_{\text{rep}}[\rho_0 + \delta \rho] \quad (20)
\]

The first and second order (so called electronic) terms are calculated by expanding the Kohn-Sham states into a linear combination of atomic orbitals (LCAO) using a small valence-only basis set. The resulting Hamiltonian is treated in a two center approximation, allowing for a very efficient construction using distance dependent pretabulated Hamiltonian and overlap integrals. By solving the resulting generalized eigenvalue problem, one obtains eigenvectors as well as eigenvalues. Since SCC-DFTB considers charge fluctuations that depend on the eigenvectors, a self-consistent treatment is necessary to obtain them.

The zeroth order (so called repulsive) term, is usually represented as a sum of atomic pair-potentials, in line with the two-center approximation in the Hamiltonian:

\[
E_{\text{rep}}(\rho_0; \{ R_{AB} \}) = \frac{1}{2} \sum_A \sum_{B \neq A} V_{AB}^{\text{rep}}(|R_A - R_B|) \quad (21)
\]

The atomic pair-potentials are fitted using ab initio or experimental reference data, in order to compensate the errors resulting from the approximate treatment of the electronic terms. This way, the DFTB method offers efficient, atomistic, quantum mechanical simulations with reasonable accuracy.

FIG. 4. Comparison between Fortnet’s analytical and numerical force analysis, in terms of force related CPU time.
B. Limitations

The restrictions imposed by the two-body representation of the repulsive energy lead in certain cases to transferability issues regarding energetics in very different chemical environments. Prominent examples include the borg-0-1\textsuperscript{47} parametrization with an inaccurate 2D-3D transition of boron clusters\textsuperscript{48}, the incorrect stability order of the ZnO wurtzite and NaCl phases\textsuperscript{49} provided by the znorg-0-1\textsuperscript{50} parameters or the silicon polymorph issue, as already observed by Chou et al.\textsuperscript{51}. To some extent, these mis-descriptions can be cured, for example by fitting to a dataset containing different polymorphs. In the ZnO case this led to a corrected stability order\textsuperscript{49} but always compromises between transferability and accuracy, resulting from the inevitable limited geometric information and flexibility of the functional foundation.

Recently, Ammothum Kandy et al.\textsuperscript{22} implemented the curvature constrained splines (CCS) framework and demonstrated that full energetic transferability between the silicon phases may be achieved by combining self-consistent DFTB with a near-sighted artificial neural network.

C. Training on Total Energy

1. Objective

Our goal is to study various different training and evaluation strategies in order to obtain DFT total energies from an ANN. We investigate and compare following scenarios:

1. Training on $E_{\text{tot}}^\text{DFT}$, the DFT total energies, enabling the ANN to predict the total energies directly.

2. Training on $E_{\text{tot}}^\text{DFT} - E_{\text{elec}}^\text{DFTB(pbc)}$, so that the ANN predicts repulsive energies on top of the electronic energies obtained from the pbc-0-3\textsuperscript{52} set. This set is known for having a good overall performance in the energetic description of silicon.

3. Training on $E_{\text{tot}}^\text{DFT} - E_{\text{elec}}^\text{DFTB(siband)}$, so that the ANN predicts repulsive energies on top of the electronic energies obtained from the band-1-1\textsuperscript{53} parameter set. This set provides a very accurate description of the silicon band structure.

4. Training on $E_{\text{tot}}^\text{DFT} - E_{\text{tot}}^\text{DFTB(pbc)}$, so that the ANN becomes capable to compensate the errors in the total energies obtained from the pbc-set. Compared to scenario 2, here the repulsive energy of the pbc-set is taken into account and serves as a kind of baseline.

5. Training on $E_{\text{tot}}^\text{DFT} - E_{\text{tot}}^\text{DFTB(siband+CCS)}$, where a CCS type\textsuperscript{22} repulsive potential fitted for theiband set serves as a baseline. (The siband set originally does not contain repulsive energy contributions.)

By covering the quantum mechanically relevant part, including long-range electrostatics, through the DFTB-Hamiltonian in approaches 2–5, the ANN is left to predict a near-sighted classical potential. With a fully quantum mechanical treatment of the electrons (at least those being designated to the valence space), we could expect to find an improved transferability to other system sizes, new chemical environments or electronic temperatures etc.

Our investigation primarily aims at addressing whether the time penalty caused by the diagonalizations of the DFTB Hamiltonian is justified by a substantially improved transferability. The aim is to compile a dataset that allows to investigate the overall accuracy of networks trained on the various energy terms presented above. The calculations examine silicon in the bulk phase as a model system. Special attention has been paid to the description of intrinsic point defects such as the vacancy or self-interstitial configurations, which are not exactly well described in the domain of the DFTB method and thus offer a great potential for improvement. The reference will be GGA-PBE\textsuperscript{23} DFT as implemented by the Vienna Ab initio Simulation Package\textsuperscript{54}, trying to reproduce the total energy and formation energy of several point defects.

In particular, once the trained networks are available, it is necessary to investigate which ones exhibit the lowest generalization error. Here, the transferability to larger supercells roughly holding eight times the atom count shall be tested. A satisfactory transferability to larger systems is of central importance, since the dataset generation would be more efficient if only moderately sized systems need to be included.

2. Dataset

The dataset holds a total number of 6306 unique homonuclear geometries with 64 ± 1 atoms per 3-dimensional periodic silicon supercell. The dataset is divided into three major categories, serving different purposes during training:

1. relaxed point defects

2. low temperature molecular dynamics

3. randomly rattled point defects

The first subset regards relaxed point defects, such as a vacancy, ⟨100⟩-split interstitial, ⟨110⟩-split interstitial, hexagonal-site interstitial or tetrahedral-site interstitial. Other defects, which should be predicted later, had been omitted from the training set to test the generalization capability of the neural network. To ensure that the included structures are assigned enough impact onto the
training, the corresponding gradients during backpropagation were homogeneously weighted by a factor of 200.

The second subset contains geometries of low temperature $T = 500 K$ bulk silicon molecular dynamic (MD) simulations. Every 50th step, the resulting geometry has been extracted, on which basis a single-point calculation was carried out to obtain the total energy terms. The resulting geometries generate a dense sampling of the nearly perfect silicon supercells, with minor deviations mostly following the normal modes.

The third subset contains randomly distorted (rattled) geometries of the relaxed point defects. Their generation utilized normal distributed random numbers to randomize the coordinates of those atoms that are directly involved in building the defect, whereby surrounding atoms were masked to remain unaltered. This certainly results in structures far away from the energetic minimum of the relaxed geometries, which actually is intentional so that the network has “seen” a wide range of fingerprints and energies during the training. For the silicon vacancy, a higher count of geometries had been included, as it was found that the network tends to have some difficulties learning the energetics of its four interacting dangling bonds. The described compilation of the training dataset is reflected by Figure 5.

Every single-point calculation was performed by GGA-PBE DFT as well as DFTB with a $\Gamma$-centered $4 \times 4 \times 4$ Monkhorst-Pack $^{55}$ k-point sampling for the 64 atom cells, and with a $2 \times 2 \times 2$ sampling for the 512 atom supercells in the test set. The DFT simulations were carried out by the Vienna Ab initio Simulation Package $^{54,56–58}$ with the GGA-PBE $^{24}$ exchange-correlation functional, plane-wave basis cutoff of 500 eV, self-consistent field tolerance of $10^{-6}$ eV and four explicit valence electrons. In analogy, the SCC-DFTB calculations were performed by the DFTB$^{+10}$ package, utilizing two established parametrizations: pbc-0-3$^{52}$ featuring a minimal s,p-basis and siband$^{53}$ with an extended basis that further includes d-orbitals.

### 3. Transferability of DFTB-Hamiltonian aided DFT Total Energy Reproduction

In this section we compare different possibilities to provide energetic corrections to the DFTB Hamiltonian with DFT total energies serving as reference. A BPNN with a [9-3-3-1] silicon sub-network and $N_{rad}=5$ radial $G^2$ functions, $N_{ang}=4$ angular $G^5$ functions and a cutoff of $R_c=5 \text{ Å}$, parametrized according to Fortnet's automatic parameter generation scheme, was chosen to carry out the investigations. A logistic, sigmoidal transfer function was deployed to calculate the neuron activations. In an early-stopping manner, the termination criterion of the training runs was provided by the arrival at the minimum prediction loss, whereas the runs were initially performed until full convergence. Having scanned the relevant iteration space, the network state of the corresponding early-stopping iteration got extracted.

Our study focuses on the transferability of the trained networks. As described above, our training set is composed of various 64 atom Si supercells. The test set, on the other hand, consists of 512 atom silicon supercells only: the perfect supercell and five supercells with various point defects. Figure 6 displays the resulting root mean square training and prediction loss values.

![FIG. 5. Contributions to the training dataset deployed to investigate the achieved accuracy when fitting a BPNN to various system-wide energy terms. The set holds a total number of 6306 unique geometries (7500 if the individual weighting of certain structures is considered) with 64 $\pm$ 1 atoms per 3-dimensional periodic supercell.](image)

![FIG. 6. Training and prediction loss when fitting a BPNN of fixed topology to various energy targets with DFT reference. The bars correspond to the objectives described in Section IV C 1.](image)
by the long-ranged defect-defect interactions. The energetic situation is qualitatively different in the larger supercells with much longer defect-defect distances, which explains the vast discrepancy between training and prediction loss.

The prediction capability of the ANN improves considerably, if the DFTB Hamiltonian is used to describe the electronic properties of the system and the ANN only provides a correction on top of it. We have followed two different strategies, by either replacing the entire repulsive contribution in DFTB with the ANN, or by using a simple “baseline” two-body repulsive potential in DFTB and correcting the resulting DFTB total energies with the ANN. As described above, both strategies have been tested with two different parametrizations, one with a good overall performance (pbc set) and one which was tuned to deliver a very precise band structure (siband set). The yellow and green bars in Figure 6 show the performance of the two sets, when the ANN replaces the entire repulsive contribution. With a prediction loss that is only roughly a factor of two higher than the training loss, these results are already satisfactory, especially considering that the prediction supercells contain eight times more atoms when compared to the training structures. If we include a two-body baseline repulsive potential (pink and blue bars in Figure 6), the results improve even further, if the the repulsive contribution is derived for a well balanced electronic parametrization, as in the case of the pbc set (blue bar in Figure 6). In this case the network solely provides minor corrections that are impossible to resolve with the limited functional form of two-center repulsive potentials. As these corrections are short-ranged, it results in a superior transferability, where the performance of the ANN, which was trained on 64±1 atom supercells, is almost unchanged when applied to 512±1 atom supercells. To us, this seems to be the most promising DFTB based machine learning workflow. However, the success of this strategy apparently needs a well balanced parametrization set. When applying it to the siband set, which was highly tuned (over-optimized) to deliver very accurate valence and conduction band edge states, the transferability is far more worse than with the balanced pbc parametrization (compare pink and blue bars in Figure 6).

Finally, we further investigated the ANN performance, when calculating the formation energies of the five point defects in the large supercell. The formation energy $E_f$ of a given defect was calculated as

$$E_f = E_{\text{tot}} - \sum_i n_i \mu_i$$

where $E_{\text{tot}}$ is the total energy of the defective system, $n_i$ the number of atoms of a given atom type $i$ in the system and $\mu_i$ the chemical potential of an atom of type $i$ in a reference system. We have chosen the perfect diamond like bulk phase of silicon as reference system. Figure 7 illustrates the results, based on the ANN potentials. Since systematic errors may cancel out in the formation energy expression, the statements go beyond a plain accuracy comparison, in the sense that networks, whose error is more systematic should perform better. The formation energies also allow us to compare the performance of the ANN with the performance of the original, uncorrected pbc set for the same defect. It should be noted, that the correct description of the energetics of such point defects has not received special attention during the creation of the pbc-0-3 silicon parametrization\cite{Eh Mayer}, consequently it performs significantly worse in predicting formation energies than its ANN corrected version, which shows the best performance. Interestingly, due to error cancellation, even the highly optimized siband set performs quite well in predicting the formation energies when corrected with the ANN.

![Diagram](image)

FIG. 7. Absolute formation energy difference for various silicon point defects and diamond bulk structure, w.r.t. GGA-PBE DFT. The bars correspond to the objectives described in Section IV C 1. Additionally, the results obtained with the original, uncorrected pbc set are also shown ($E_{\text{DFTB(pbc)}}$).

V. SUMMARY

We have implemented a parallel Behler-Parrinello neural network in the open source Fortnet software package. The software package offers a wide variety of features, including a truncated Xavier network parameter initialization (primarily intended to operate in combination with a sigmoidal activation function), the capability to train on atomic or system-wide properties by using different loss functions and regularization techniques as well as an atomic force analysis. The parallel efficiency of the implementation was demonstrated by a realistic benchmark. Further key extensions are planned in the future, such as support for a mixture of atomic and system-wide target properties and library interfacing.

Using the Fortnet software package, we have explored various possibilities to combine ANNs with the DFTB method in order to obtain DFT total energies and formation energies. We have demonstrated, that by training an ANN to predict the difference between the total energies calculated by DFT and DFTB, one can achieve near
DFT-accuracy, when calculating the formation energies of some selected point defects in silicon, at the computational cost of some much cheaper DFTB method. By training our ANN on small supercells only and subsequently evaluating its performance on significantly larger supercells, we could demonstrate that in contrast to the direct prediction of the DFT energies by an ANN, the combined DFTB+ANN approach features a significantly improved transferability. Although the computational cost of this combined method is higher as if only a pure ANN had to be evaluated, its superior transferability makes it nevertheless a promising approach for materials simulations. So far, our research was limited to single-point calculations, future studies will therefore plan to target molecular dynamic simulations.

SUPPLEMENTARY MATERIAL

See supplementary material for exemplary Human-friendly Structured Data (HSD) input listings, as well as the basic usage of the Fortformat Python layer for generating datasets and extracting results.

ACKNOWLEDGMENTS

We would like to acknowledge the Swedish e-science initiative eSSENCE, the Swedish national infrastructure for computing (SNIC), and the Swedish research council (VR). The simulations were performed on the HPC cluster Aether at the University of Bremen, financed by DFG within the scope of the Excellence Initiative.

DATA AVAILABILITY STATEMENT

All datasets employed to obtain application related findings of this study are openly available at https://doi.org/10.5281/zenodo.5969907.

Appendix A: Analytical Expression for ASCF Derivatives

The derivatives of the radial ACSFs w.r.t atomic positions $R$ are given by:

$$
\frac{\partial G_i^{1, Z_i}}{\partial R^{\alpha}_{k \neq i}} = f_{c,i,k} \hat{R}^{\alpha}_{ik}
$$

(A1a)

$$
\frac{\partial G_i^{2, Z_i}}{\partial R^{\alpha}_{k \neq i}} = \left(2 \eta(R_s - R_{ij}) f_{c,i,k} + f'_{c,i,k}\right) e^{-\eta(R_{ij} - R_s)^2} \hat{R}^{\alpha}_{ik}
$$

(A1b)

$$
\frac{\partial G_i^{3, Z_i}}{\partial R^{\alpha}_{k \neq i}} = \left(- \kappa \sin(\kappa R_{ik}) f_{c,i,k} + \cos(\kappa R_{ik}) f'_{c,i,k}\right) \hat{R}^{\alpha}_{ik}
$$

(A1c)

Here, $\hat{R}^{\mu}_{ab}$ is the $\mu$-th component of the unit vector from atom $a$ to atom $b$ and $R_{ab}$ is the distance between them. Moreover, $f_{c,ab} \equiv f_c(R_{ab})$, where $f_c$ is the cutoff function introduced by eq. (8) and $f'_c$ its first derivative:

$$
f'_c(r) = -\frac{\pi}{2R_c} \sin\left(\frac{\pi r}{R_c}\right)
$$

(A2)

The derivative of the $G^4$ function w.r.t. $R^{\mu}_{k}$ reads:

$$
\frac{\partial G_i^{4, Z_i} Z_j}{\partial R^{\mu}_{k \neq i}} = 2^{1-\xi} \delta_{Z_k, Z_i} \times \left\{ \sum_{j \neq i} |Z_j| f_{cij} A_{\lambda} (\lambda) \left[ \lambda_f f_{cij} \hat{R}^{\alpha}_{ik} \sum_{\beta} T^{\alpha \beta}_{ik} \hat{R}^{\beta}_{ij} + A_{ij}(\lambda) \left( f'_{cij} + f_{cij} \hat{R}^{\alpha}_{ik} \hat{R}^{\alpha}_{jk} \right) \right] \right\}
$$

$$
+ 2^{1-\xi} \delta_{Z_k, Z_2} \left\{ Z_2 \rightarrow Z_1 \right\},
$$

(A3)

Here, $\{ Z_2 \rightarrow Z_1 \}$ indicates that the preceding expression in curly brackets is to be repeated, with $Z_2$ substituted by $Z_1$, $f_{cij}(r) \equiv e^{-\eta r^2} f_c(r)$ is the exponentially damped cut-off function ($f_{cij} \equiv f_c(R_{ab})$), while $A_{ij}(\lambda)$ and $T^{\mu \nu}_{ab}$ are defined as:

$$
A_{ij}(\lambda) \equiv 1 + \lambda \hat{R}^{\alpha}_{ij} \hat{R}^{\alpha}_{ik}
$$

(A4a)

$$
T^{\mu \nu}_{ab} \equiv \frac{1}{R_{ab}} \left( \delta_{\mu \nu} - \hat{R}^{\mu}_{ab} \hat{R}^{\nu}_{ab} \right)
$$

(A4b)

The formula for the derivative of the $G^5$ function can be obtained directly from (A3) by treating $f_{cij}$ as a constant equal to one. Finally, all the preceding expressions are valid for taking the derivatives of the $G_i$ functions w.r.t. positions of other atoms (i.e., whose index is not $i$). Due to the translational invariance, the derivative of any $G_i$-function w.r.t. the position of atom $i$ can be obtained from its derivatives w.r.t. other atoms, following the expression below:

$$
\frac{\partial G_i}{\partial R^{\alpha}_{k \neq i}} = - \sum_{j \neq i} \frac{\partial G_i}{\partial R^{\alpha}_{j \neq i}}
$$

(A5)

1W. S. McCulloch and W. Pitts, The bulletin of mathematical biophysics 5, 115 (1943).
2K. Hornik, M. Stinchcombe, and H. White, Neural Networks 2, 359 (1989).
3M. Egmont-Petersen, D. de Ridder, and H. Handels, Lect Notes Comput Sc 35, 2279 (2002).
4G. E. Dahl, D. Yu, L. Deng, and A. Acero, IEEE Transactions on Audio, Speech, and Language Processing 20, 30 (2012).
5T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, in 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2015) pp. 4580–4584.
6E. Trentin and M. Gori, Neurocomputing 37, 91 (2001).
7J. Behler and M. Parrinello, Phys. Rev. Lett. 98, 144601 (2007).
8J. Goodfellow, Y. Bengio, and A. Courville, Deep Learning (MIT Press, 2016).
