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

# GPUMD: A package for constructing accurate machine-learned potentials and performing highly efficient atomistic simulations

Cite as: J. Chem. Phys. 157, 114801 (2022); doi: 10.1063/5.0106617
Submitted: 29 June 2022 • Accepted: 24 August 2022 •
Published Online: 20 September 2022

View Online Export Citation CrossMark

Zheyong Fan,[1,a] Yanzhou Wang,[2,3] Penghua Ying,[4] Keke Song,[3] Junjie Wang,[5] Yong Wang,[5] Zezhu Zeng,[6] Ke Xu,[7] Eric Lindgren,[8] J. Magnus Rahm,[8] Alexander J. Gabourie,[9] Jiahui Liu,[3] Haikuan Dong,[1,3] Jianyang Wu,[7] Yue Chen,[6] Zheng Zhong,[4] Jian Sun,[5,b] Paul Erhart,[8,c] Yanjing Su,[3,d] and Tapio Ala-Nissila[2,10]

## AFFILIATIONS

[1]College of Physical Science and Technology, Bohai University, Jinzhou 121013, People's Republic of China
[2]MSP Group, QTF Centre of Excellence, Department of Applied Physics, Aalto University, FI-00076 Aalto, Espoo, Finland
[3]Beijing Advanced Innovation Center for Materials Genome Engineering, University of Science and Technology Beijing, Beijing 100083, China
[4]School of Science, Harbin Institute of Technology, Shenzhen 518055, People's Republic of China
[5]National Laboratory of Solid State Microstructures, School of Physics and Collaborative Innovation Center of Advanced Microstructures, Nanjing University, Nanjing 210093, China
[6]Department of Mechanical Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong SAR, China
[7]Department of Physics, Research Institute for Biomimetics and Soft Matter, Jiujiang Research Institute and Fujian Provincial Key Laboratory for Soft Functional Materials Research, Xiamen University, Xiamen 361005, People's Republic of China
[8]Chalmers University of Technology, Department of Physics, 41926 Gothenburg, Sweden
[9]Department of Electrical Engineering, Stanford University, Stanford, California 94305, USA
[10]Interdisciplinary Centre for Mathematical Modelling, Department of Mathematical Sciences, Loughborough University, Loughborough, Leicestershire LE11 3TU, United Kingdom

**Note:** This paper is part of the JCP Special Topic on Software for Atomistic Machine Learning.
[a]**Author to whom correspondence should be addressed:** brucenju@gmail.com
[b]**Electronic mail:** jiansun@nju.edu.cn
[c]**Electronic mail:** erhart@chalmers.se
[d]**Electronic mail:** yjsu@ustb.edu.cn

## ABSTRACT

We present our latest advancements of machine-learned potentials (MLPs) based on the neuroevolution potential (NEP) framework introduced in Fan *et al.* [Phys. Rev. B **104**, 104309 (2021)] and their implementation in the open-source package GPUMD. We increase the accuracy of NEP models both by improving the radial functions in the atomic-environment descriptor using a linear combination of Chebyshev basis functions and by extending the angular descriptor with some four-body and five-body contributions as in the atomic cluster expansion approach. We also detail our efficient implementation of the NEP approach in graphics processing units as well as our workflow for the construction of NEP models and demonstrate their application in large-scale atomistic simulations. By comparing to state-of-the-art MLPs, we show that the NEP approach not only achieves above-average accuracy but also is far more computationally efficient. These results demonstrate that the GPUMD package is a promising tool for solving challenging problems requiring highly accurate, large-scale atomistic simulations. To enable the construction of MLPs using a minimal training set, we propose an active-learning scheme based on the latent


Published under an exclusive license by AIP Publishing

space of a pre-trained NEP model. Finally, we introduce three separate Python packages, viz., GPYUMD, CALORINE, and PYNEP, that enable the integration of GPUMD into Python workflows.

*Published under an exclusive license by AIP Publishing.* https://doi.org/10.1063/5.0106617

## I. INTRODUCTION

Machine-learned classical potentials[1–7] have shown great promise in enabling accurate atomistic simulations far beyond the space and time scales that can be achieved using quantum-mechanical calculations. Many open-source computer packages for machine-learned potentials (MLPs) have been published, including QUIP-GAP,[8–10] SNAP,[11] AMP,[12] AENET,[13,14] ANI,[15] SCHNET,[16] DEEPMD-KIT,[17–19] TENSORMOL,[20] PHYSNET,[21] MEGNET,[22] TURBOGAP,[23] SGDML,[24] N2P2,[25] SIMPLE-NN,[26] PANNA,[27] FCHL,[28] PINN,[29] MLIP,[30,31] REANN,[32] TABGAP,[33] PYXTAL-FF,[34] PYTHON-ACE,[35,36] and KLIFF.[37] However, most existing implementations of MLPs (TABGAP[33] is a notable exception) have a computational speed of about $1 \times 10^3$ atom step/s using one typical central processing unit (CPU) core, which is about two to three orders of magnitude slower than typical empirical potentials such as the Tersoff potential.[38] Therefore, even though MLPs are already faster than quantum-mechanical methods, it is still desirable to speed up MLPs as much as possible.

One approach to speed up MLPs is to use a huge number of CPUs and/or graphics processing units (GPUs) through message-passing information parallelization. For example, using 27 360 V100 GPUs and the same number of CPU cores, the deep potential (DP) approach[17–19] has been used to simulate a 127-million-atom aluminum system with a speed of about $1.23 \times 10^9$ atom step/s.[39] However, such huge amounts of computational resources are not available to most researchers. More importantly, the performance of DP per V100 GPU is only about $4.5 \times 10^4$ atom step/s, which is about ten times *slower* than an empirical embedded atom method (EAM) potential[40] with a single typical CPU core.

A more economical approach is to optimize the formulation and implementation of the MLP itself so that it can attain a high computational speed using a reasonable amount of computational resources available to most researchers. To this end, we have developed a MLP called neuroevolution potential (NEP)[41,42] within the GPUMD package[43,44] that can achieve a computational speed of about $1 \times 10^7$ atom step/s using a single V100 GPU, which is about ten times *faster* than the empirical Tersoff potential on a single typical CPU core.

In this paper, we present recent developments of the NEP approach that further improve its accuracy without reducing the efficiency. Specifically, we improve the radial functions in the atomic-environment descriptor by using a combination of basis functions, and add angular descriptor components with high-order correlations. The improved radial functions are better at distinguishing different atom types and lead to higher accuracy in multicomponent systems. The added angular descriptor components with high-order correlations make the atomic-environment descriptor more complete and help to increase the regression accuracy.

Using a number of systems, including MgAlCu alloy, silicon with various phases, the azobenzene molecule, and carbon with various phases, we demonstrate the accuracy and efficiency of the latest NEP as implemented in the GPUMD package. We compare with other state-of-the-art MLPs, including DP,[17–19] Gaussian approximation potential (GAP),[8] moment tensor potential (MTP),[30,31] recursive embedded-atom neural network (REANN),[32,45] and atomic cluster expansion (ACE).[46–48] Through these comprehensive comparisons, we show that the NEP implementation in GPUMD can achieve a computational speed that is far superior to other MLPs, under the condition of achieving an above-average accuracy. We present the algorithms for the efficient GPU implementation of NEP in great detail. Using a single GPU, such as an A100, one can use GPUMD to simulate up to $10 \times 10^6$ atoms on nanosecond time scales, which can only be achieved by using a huge amount of computational resources with other publicly available codes. The GPUMD package makes large-scale, high-accuracy atomistic simulations available to a wide community instead of only a small number of institutions. In addition to these high-efficiency atomistic simulations, we also propose an effective active-learning scheme based on the latent space of the NEP model that can greatly reduce the computational burden of preparing training data.

Finally, we introduce the workflow for constructing and using NEPs through concrete examples in atomistic simulations of various materials properties, including lattice constant, elastic constants, stress–strain relation during tensile loading, structural properties during a melt–quench–anneal process, diffusion coefficient, and thermal properties of amorphous structures. We also describe interfacing GPUMD to Python via the GPYUMD, CALORINE, and PYNEP packages.

## II. THEORETICAL FORMULATIONS OF THE NEP APPROACH

The first NEP, called NEP1, was proposed in Ref. 41. An improved version, called NEP2, is presented in Ref. 42. In the present paper, we further refine the NEP approach and introduce NEP3. In this section, we present NEP3 and discuss the differences to NEP1 and NEP2.

### A. The neural network model

Following Behler and Parrinello,[49] the site energy of atom $i$ is taken as a function of the descriptor vector with $N_{\rm des}$ components, $U_i(\mathbf{q}) = U_i\left(\{q_\nu^i\}_{\nu=1}^{N_{\rm des}}\right)$. We use a feedforward neural network with a single hidden layer with $N_{\rm neu}$ neurons to model this function,

$$U_i = \sum_{\mu=1}^{N_{\rm neu}} w_\mu^{(1)} \tanh\left(\sum_{\nu=1}^{N_{\rm des}} w_{\mu\nu}^{(0)} q_\nu^i - b_\mu^{(0)}\right) - b^{(1)}, \tag{1}$$

where $\tanh(x)$ is the activation function in the hidden layer, $\mathbf{w}^{(0)}$ is the connection weight matrix from the input layer (descriptor


Published under an exclusive license by AIP Publishing

vector) to the hidden layer, $\mathbf{w}^{(1)}$ is the connection weight vector from the hidden layer to the output node, which is the energy $U_i$, $\mathbf{b}^{(0)}$ is the bias vector in the hidden layer, and $b^{(1)}$ is the bias for node $U_i$. The total number of parameters in the neural network is thus $(N_{\rm des}+2)N_{\rm neu}+1$. The descriptor vector is formed by juxtaposition of a number of components, including those with radial (distance) information only, which are called radial descriptor components, and those with both radial and angular information, which are called angular descriptor components. The descriptor is one of the most important aspects in MLPs.[50,51] We discuss the radial and angular descriptor components below.

### B. Radial descriptor components

There are $n^{\rm R}_{\rm max}+1$ radial descriptor components and they are defined as

$$q^i_n = \sum_{j\neq i} g_n(r_{ij}) \quad \text{with} \quad 0 \le n \le n^{\rm R}_{\rm max}, \tag{2}$$

where the summation runs over all the neighbors of atom $i$ within a certain cutoff distance. The functions $g_n(r_{ij})$ depend on the distance $r_{ij}$ only and are, therefore, called the radial functions. In NEP3, they are defined as a linear combination of $N^{\rm R}_{\rm bas}+1$ basis functions $\{f_k(r_{ij})\}_{k=0}^{N^{\rm R}_{\rm bas}}$,

$$g_n(r_{ij}) = \sum_{k=0}^{N^{\rm R}_{\rm bas}} c^{ij}_{nk} f_k(r_{ij}), \tag{3}$$

with

$$f_k(r_{ij}) = \frac{1}{2}\Big[T_k\Big(2\big(r_{ij}/r^{\rm R}_{\rm c}-1\big)^2-1\Big)+1\Big]f_{\rm c}(r_{ij}). \tag{4}$$

Both $n^{\rm R}_{\rm max}$ and $N^{\rm R}_{\rm bas}$ are tunable hyperparameters in NEP3, which, along with other ones, will be listed in Sec. II H 2. In Sec. III B, we will use a few examples to illustrate the judicious choice of the various hyperparameters in typical applications. Here, $T_k(x)$ is the $k$th-order Chebyshev polynomial of the first kind and $f_{\rm c}(r_{ij})$ is the cutoff function defined as

$$f_{\rm c}(r_{ij}) = \begin{cases} \frac{1}{2}\Big[1+\cos\Big(\pi\frac{r_{ij}}{r^{\rm R}_{\rm c}}\Big)\Big], & r_{ij} \le r^{\rm R}_{\rm c}; \\ 0, & r_{ij} > r^{\rm R}_{\rm c}. \end{cases} \tag{5}$$

Here, $r^{\rm R}_{\rm c}$ is the cutoff distance of the radial descriptor components. The expansion coefficients $c^{ij}_{nk}$ depend on $n$ and $k$ and also on the types of atoms $i$ and $j$. Due to the summation over neighbors, the radial descriptor components defined above are invariant with respect to permutation of atoms of the same type.

If the material considered has $N_{\rm typ}$ atom types, the number of $c^{ij}_{nk}$ coefficients for the radial descriptor components is $N^2_{\rm typ}\big(n^{\rm R}_{\rm max}+1\big)\big(N^{\rm R}_{\rm bas}+1\big)$. In NEP2, each radial function is simply a basis function and $c^{ij}_{nk}$ is reduced to $\delta_{nk}c_{nij}$, where $\delta_{nk}$ is the Kronecker symbol and $c_{nij}$ is defined in Ref. 42. In NEP2 and NEP3, these coefficients are trainable (similar to the parameters in the neural network), while in NEP1, we have used fixed coefficients similar to previous works.[14,52] In Ref. 42, we have shown that NEP2 is much more accurate than NEP1 for multicomponent systems. In this paper, we will show that the accuracy of NEP3 for multicomponent systems is further improved as compared to NEP2.

### C. Angular descriptor components

In NEP1 and NEP2, the angular descriptor components $\{q^i_{nl}\}$ are taken as ($0 \le n \le n^{\rm A}_{\rm max}$ and $1 \le l \le l^{\rm 3b}_{\rm max}$) follows:

$$q^i_{nl} = \frac{2l+1}{4\pi}\sum_{j\neq i}\sum_{k\neq i} g_n(r_{ij})g_n(r_{ik})P_l(\cos\theta_{ijk}), \tag{6}$$

where $P_l(\cos\theta_{ijk})$ is the Legendre polynomial of order $l$ and $\theta_{ijk}$ is the angle formed by the $ij$ and $ik$ bonds. The radial functions $g_n(r_{ij})$ have the same forms as in Eq. (3), but they can have a different cutoff distance $r^{\rm A}_{\rm c}$ and a different basis size $N^{\rm A}_{\rm bas}$ than those in the radial descriptor components. Usually, it is beneficial to use $r^{\rm A}_{\rm c} < r^{\rm R}_{\rm c}$, assuming that there is no directional dependence of the descriptor on some neighboring atoms that are sufficiently far away from the central atom. This reflects the physical intuition that interaction strength decreases both with distance and order. The radial descriptor components are relatively cheap to evaluate and one can, thus, use a relatively large radial cutoff distance $r^{\rm R}_{\rm c}$ (also relatively large $n^{\rm R}_{\rm max}$ and $N^{\rm R}_{\rm bas}$) combined with a relatively small angular cutoff distance $r^{\rm A}_{\rm c}$ (also relatively small $n^{\rm A}_{\rm max}$ and $N^{\rm A}_{\rm bas}$) to achieve a good balance between accuracy and speed. Note that message-passing could effectively increase the interaction range but is not necessarily an efficient way of describing long-range interactions.[53] Using a relatively large cutoff for the radial descriptor components is generally a more efficient way of incorporating long-range interactions, such as van der Waals (vdW) and screened Coulomb interactions, although it is incapable of describing genuine long-range interactions such as unscreened Coulomb interactions.

Expression (6) is not efficient for numerical evaluation due to the double summation over neighbors. An equivalent form that is more efficient for numerical evaluation can be obtained by using the addition theorem of the spherical harmonics as

$$q^i_{nl} = \sum_{m=-l}^{l}(-1)^m A^i_{nlm}A^i_{nl(-m)} = \sum_{m=0}^{l}(2-\delta_{0l})|A^i_{nlm}|^2, \tag{7}$$

where

$$A^i_{nlm} = \sum_{j\neq i} g_n(r_{ij})Y_{lm}(\theta_{ij},\phi_{ij}). \tag{8}$$

Here, $Y_{lm}(\theta_{ij},\phi_{ij})$ are the spherical harmonics as a function of the polar angle $\theta_{ij}$ and the azimuthal angle $\phi_{ij}$ for the position difference vector $\boldsymbol{r}_{ij} \equiv \boldsymbol{r}_j - \boldsymbol{r}_i$ from atom $i$ to atom $j$. In Eq. (7), we have used the property $A_{nl(-m)} = (-1)^m A^*_{nlm}$, which follows from the property $Y_{l(-m)} = (-1)^m Y^*_{lm}$.


Published under an exclusive license by AIP Publishing

The angular descriptor components above are usually known as three-body ones as in the ACE approach,[46] although all the descriptor components are many-body in nature. For simplicity, we will use the ACE terminology. Higher-order angular descriptor components can be similarly constructed.[46] In NEP3, we add the following four-body descriptor components ($1 \le l_1 \le l_2 \le l_3 \le l^{4b}_{\max}$):

$$q^i_{nl_1l_2l_3} = \sum_{m_1=-l_1}^{l_1} \sum_{m_2=-l_2}^{l_2} \sum_{m_3=-l_3}^{l_3} \begin{pmatrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \end{pmatrix} A^i_{nl_1m_1} A^i_{nl_2m_2} A^i_{nl_3m_3}, \quad (9)$$

and the following five-body ones ($1 \le l_1 \le l_2 \le l_3 \le l_4 \le l^{5b}_{\max}$):

$$q^i_{nl_1l_2l_3l_4} = \sum_{m_1=-l_1}^{l_1} \sum_{m_2=-l_2}^{l_2} \sum_{m_3=-l_3}^{l_3} \sum_{m_4=-l_4}^{l_4} \begin{bmatrix} l_1 & l_2 & l_3 & l_4 \\ m_1 & m_2 & m_3 & m_4 \end{bmatrix} \times A^i_{nl_1m_1} A^i_{nl_2m_2} A^i_{nl_3m_3} A^i_{nl_4m_4}, \quad (10)$$

where $\begin{pmatrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \end{pmatrix}$ are Wigner 3$j$ symbols and[46]

$$\begin{bmatrix} l_1 & l_2 & l_3 & l_4 \\ m_1 & m_2 & m_3 & m_4 \end{bmatrix} = \sum_{L=\max\{|l_1-l_2|,|l_3-l_4|\}}^{\min\{|l_1+l_2|,|l_3+l_4|\}} \sum_{M=-L}^{L} (-1)^M \times \begin{pmatrix} L & l_1 & l_2 \\ -M & m_1 & m_2 \end{pmatrix} \begin{pmatrix} L & l_3 & l_4 \\ M & m_3 & m_4 \end{pmatrix}. \quad (11)$$

We can consider higher-order terms,[46] but to keep a balance between accuracy and speed, we only consider those up to fifth order. Recent work suggests that four-body interactions are difficult to learn using only three-body correlations[54] and that, using only up to four-body correlations, they can still yield identical results for different configurations of simple molecules.[55] One can formally achieve completeness in the MTP formalism,[30] the ACE formalism,[46,48] and other related ones,[56–58] but in practice, there is always a truncation of the descriptor size and a balance must be struck between accuracy and speed. This balance is one of the most important guidelines for the development of NEP in GPUMD.

### D. Explicit expressions for the angular descriptor components

The angular descriptor components are quite complicated and care must be taken to achieve an efficient implementation. There have been some implementations of the ACE approach combined with linear regression, where it has been found that recursive evaluation can lead to much higher efficiency.[35,36,48] In our GPU implementation of NEP3 with the ACE-like descriptor components, we find it crucial to derive the relevant expressions as explicitly as possible as it allows us to reduce the number of terms to be evaluated, thanks to symmetry considerations.

To facilitate the following presentation, we define a series of summations that are used to express the angular descriptor components ($0 \le n \le n^{A}_{\max}$ and $0 \le k \le 23$),

$$S_{n,k} = \sum_{j \neq i} \frac{g_n(r_{ij})}{r^n_{ij}} b_k(x_{ij}, y_{ij}, z_{ij}). \quad (12)$$

The functions $b_k(x_{ij}, y_{ij}, z_{ij})$ here are $z_{ij}$, $x_{ij}$, $y_{ij}$, $3z^2_{ij} - r^2_{ij}$, $x_{ij}z_{ij}$, $y_{ij}z_{ij}$, $x^2_{ij} - y^2_{ij}$, $2x_{ij}y_{ij}$, $(5z^2_{ij} - 3r^2_{ij})z_{ij}$, $(5z^2_{ij} - r^2_{ij})x_{ij}$, $(5z^2_{ij} - r^2_{ij})y_{ij}$, $(x^2_{ij} - y^2_{ij})z_{ij}$, $2x_{ij}y_{ij}z_{ij}$, $(x^2_{ij} - 3y^2_{ij})x_{ij}$, $(3x^2_{ij} - y^2_{ij})y_{ij}$, $(35z^2_{ij} - 30r^2_{ij})z^2_{ij} + 3r^4_{ij}$, $(7z^2_{ij} - 3r^2_{ij})x_{ij}z_{ij}$, $(7z^2_{ij} - 3r^2_{ij})y_{ij}z_{ij}$, $(7z^2_{ij} - r^2_{ij})(x^2_{ij} - y^2_{ij})$, $(7z^2_{ij} - r^2_{ij})2x_{ij}y_{ij}$, $(x^2_{ij} - 3y^2_{ij})x_{ij}z_{ij}$, $(3x^2_{ij} - y^2_{ij})y_{ij}z_{ij}$, $(x^2_{ij} - y^2_{ij})^2 - 4x^2_{ij}y^2_{ij}$, and $4(x^2_{ij} - y^2_{ij})x_{ij}y_{ij}$ from $k = 0$ to $k = 23$. With these summations, we can write the three-body angular descriptor components up to $l^{3b}_{\max} = 4$ explicitly as

$$q^i_{n1} = \frac{1}{4}\frac{3}{\pi} S^2_{n,0} + 2\frac{1}{4}\frac{3}{2\pi}(S^2_{n,1} + S^2_{n,2}) \equiv \sum_{k=0}^{2} C^{3b}_k S^2_{n,k}, \quad (13)$$

$$q^i_{n2} = \frac{1}{16}\frac{5}{\pi} S^2_{n,3} + 2\frac{1}{4}\frac{15}{2\pi}(S^2_{n,4} + S^2_{n,5}) + 2\frac{1}{16}\frac{15}{2\pi}(S^2_{n,6} + S^2_{n,7}) \equiv \sum_{k=3}^{7} C^{3b}_k S^2_{n,k}, \quad (14)$$

$$q^i_{n3} = \frac{1}{16}\frac{7}{\pi} S^2_{n,8} + 2\frac{1}{64}\frac{21}{\pi}(S^2_{n,9} + S^2_{n,10}) + 2\frac{1}{16}\frac{105}{2\pi}(S^2_{n,11} + S^2_{n,12}) + 2\frac{1}{64}\frac{35}{\pi}(S^2_{n,13} + S^2_{n,14}) \equiv \sum_{k=8}^{14} C^{3b}_k S^2_{n,k}; \quad (15)$$

$$q^i_{n4} = \frac{9}{256}\frac{1}{\pi} S^2_{n,15} + 2\frac{9}{64}\frac{5}{\pi}(S^2_{n,16} + S^2_{n,17}) + 2\frac{9}{64}\frac{5}{2\pi}(S^2_{n,18} + S^2_{n,19}) + 2\frac{9}{64}\frac{35}{\pi}(S^2_{n,20} + S^2_{n,21}) + 2\frac{9}{256}\frac{35}{2\pi}(S^2_{n,22} + S^2_{n,23}) \equiv \sum_{k=15}^{23} C^{3b}_k S^2_{n,k}, \quad (16)$$

whereby we defined the three-body coefficients $\{C^{3b}_k\}^{23}_{k=0}$.

For four-body angular descriptor components, we only consider the case of $l_1 = l_2 = l_3$ and up to $l^{4b}_{\max} = 2$. It turns out that $q^i_{n111} = q^i_{n333} = 0$. Therefore, there is no difference between $l^{4b}_{\max} = 2$ and $l^{4b}_{\max} = 3$. Then, we only have the case of $l_1 = l_2 = l_3 = 2$,

$$q^i_{n222} = -\sqrt{\frac{2}{35}} A^i_{n20} A^i_{n20} A^i_{n20} + 6\sqrt{\frac{1}{70}} A^i_{n20} A^i_{n21} A^i_{n2(-1)} + 6\sqrt{\frac{2}{35}} A^i_{n20} A^i_{n22} A^i_{n2(-2)} - 6\sqrt{\frac{3}{35}} A^i_{n21} A^i_{n21} A^i_{n2(-2)}. \quad (17)$$

The root-rational-fraction package[59] has been used to obtain analytical expressions of the various Wigner 3$j$ symbols. With some algebra, we have


Published under an exclusive license by AIP Publishing

$$
\begin{aligned}
q^i_{n222} = &-\sqrt{\frac{2}{35}}\left(\frac{1}{4}\sqrt{\frac{5}{\pi}}\right)^3 S^3_{n,3} - 6\sqrt{\frac{1}{70}}\left(\frac{1}{4}\sqrt{\frac{5}{\pi}}\right)\left(\frac{1}{2}\sqrt{\frac{15}{2\pi}}\right)^2 \\
&\times S_{n,3}\left(S^2_{n,4}+S^2_{n,5}\right) + 6\sqrt{\frac{2}{35}}\left(\frac{1}{4}\sqrt{\frac{5}{\pi}}\right)\left(\frac{1}{4}\sqrt{\frac{15}{2\pi}}\right)^2 \\
&\times S_{n,3}\left(S^2_{n,6}+S^2_{n,7}\right) + 6\sqrt{\frac{3}{35}}\left(\frac{1}{2}\sqrt{\frac{15}{2\pi}}\right)^2\left(\frac{1}{4}\sqrt{\frac{15}{2\pi}}\right) \\
&\times S_{n,6}\left(S^2_{n,5}-S^2_{n,4}\right) - 12\sqrt{\frac{3}{35}}\left(\frac{1}{2}\sqrt{\frac{15}{2\pi}}\right)^2\left(\frac{1}{4}\sqrt{\frac{15}{2\pi}}\right) \\
&\times S_{n,4}S_{n,5}S_{n,7} \\
\equiv\ & C^{4b}_0 S^3_{n,3} + C^{4b}_1 S_{n,3}\left(S^2_{n,4}+S^2_{n,5}\right) + C^{4b}_2 S_{n,3}\left(S^2_{n,6}+S^2_{n,7}\right) \\
&+ C^{4b}_3 S_{n,6}\left(S^2_{n,5}-S^2_{n,4}\right) + C^{4b}_4 S_{n,4}S_{n,5}S_{n,7},
\end{aligned} \tag{18}
$$

whereby we defined the four-body coefficients $\{C^{4b}_k\}^4_{k=0}$.

For five-body descriptors, we only consider up to $l_1 = l_2 = l_3 = l_4 = 1$,

$$
\begin{aligned}
q^i_{n1111} &= \frac{7}{15}\left(A^i_{n10}\right)^4 - \frac{28}{15}\left(A^i_{n10}\right)^2 A^i_{n11}A^i_{n1(-1)} + \frac{28}{15}\left(A^i_{n11}\right)^2\left(A^i_{n1(-1)}\right)^2 \\
&= \frac{21}{80\pi^2}S^4_{n,0} + \frac{21}{40\pi^2}S^2_{n,0}\left(S^2_{n,1}+S^2_{n,2}\right) + \frac{21}{80\pi^2}\left(S^2_{n,1}+S^2_{n,2}\right)^2 \\
&\equiv C^{5b}_0 S^4_{n,0} + C^{5b}_1 S^2_{n,0}\left(S^2_{n,1}+S^2_{n,2}\right) + C^{5b}_2\left(S^2_{n,1}+S^2_{n,2}\right)^2,
\end{aligned} \tag{19}
$$

whereby we defined the five-body coefficients $\{C^{5b}_k\}^2_{k=0}$.

In our implementation, the three-body coefficients $\{C^{3b}_k\}^{23}_{k=0}$, four-body coefficients $\{C^{4b}_k\}^4_{k=0}$, and five-body coefficients $\{C^{5b}_k\}^2_{k=0}$ are pre-computed. This is crucial for obtaining high computational performance.

We can now enumerate the descriptor vector length. There are $(n^R_{max}+1)$ radial descriptor components, $(n^A_{max}+1)l^{3b}_{max}$ three-body descriptor components, $(n^A_{max}+1)$ four-body descriptor components, and $(n^A_{max}+1)$ five-body descriptor components. Therefore, we have

$$
N_{des} = \left(n^R_{max}+1\right) + \left(n^A_{max}+1\right)\left(l^{3b}_{max}+2\right) \tag{20}
$$

in NEP3 if we include both the four-body and the five-body descriptor components.

## E. Force, virial, and heat current expressions

As stressed in Ref. 44, we need to derive an explicit expression of the partial force[60] for an efficient GPU implementation.

The partial force is

$$
\begin{aligned}
\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}} = &\sum_{n=0}^{n^R_{max}} \frac{\partial U_i}{\partial q^i_n}\frac{\partial q^i_n}{\partial \boldsymbol{r}_{ij}} + \sum_{n=0}^{n^A_{max}}\sum_{l=1}^{l^{3b}_{max}} \frac{\partial U_i}{\partial q^i_{nl}}\frac{\partial q^i_{nl}}{\partial \boldsymbol{r}_{ij}} \\
&+ \sum_{n=0}^{n^A_{max}}\sum_{l=1}^{l^{4b}_{max}} \frac{\partial U_i}{\partial q^i_{nlll}}\frac{\partial q^i_{nlll}}{\partial \boldsymbol{r}_{ij}} + \sum_{n=0}^{n^A_{max}}\sum_{l=1}^{l^{5b}_{max}} \frac{\partial U_i}{\partial q^i_{nllll}}\frac{\partial q^i_{nllll}}{\partial \boldsymbol{r}_{ij}}.
\end{aligned} \tag{21}
$$

Because all the relevant functions here are analytical, it is straightforward to derive explicit expressions for all the partial derivatives in the equation above.

With the partial force available, the total force acting on atom $i$ from atom $j$ can be computed as[60]

$$
\boldsymbol{F}_{ij} = \frac{\partial U_i}{\partial \boldsymbol{r}_{ij}} - \frac{\partial U_j}{\partial \boldsymbol{r}_{ji}}, \tag{22}
$$

which respects Newton's third law, $\boldsymbol{F}_{ij} = -\boldsymbol{F}_{ji}$. The total force acting on atom $i$ from all the neighboring atoms is, thus,

$$
\boldsymbol{F}_i = \sum_{j\neq i} \boldsymbol{F}_{ij}. \tag{23}
$$

From the partial force, one can define the per-atom virial,[60,61]

$$
\mathbf{W}_i = \sum_{j\neq i} \boldsymbol{r}_{ij} \otimes \frac{\partial U_j}{\partial \boldsymbol{r}_{ji}}. \tag{24}
$$

By contracting the per-atom virial above with the velocity $\boldsymbol{v}_i$, one can then obtain the per-atom heat current,[60,61]

$$
\boldsymbol{J}_i = \mathbf{W}_i \cdot \boldsymbol{v}_i = \sum_{j\neq i}\left(\boldsymbol{r}_{ij} \otimes \frac{\partial U_j}{\partial \boldsymbol{r}_{ji}}\right)\cdot \boldsymbol{v}_i = \sum_{j\neq i}\boldsymbol{r}_{ij}\left(\frac{\partial U_j}{\partial \boldsymbol{r}_{ji}}\cdot \boldsymbol{v}_i\right). \tag{25}
$$

The total heat current in the system is the sum of the per-atom contributions,

$$
\boldsymbol{J} = \sum_i \boldsymbol{J}_i = \sum_i\sum_{j\neq i}\boldsymbol{r}_{ij}\left(\frac{\partial U_j}{\partial \boldsymbol{r}_{ji}}\cdot \boldsymbol{v}_i\right). \tag{26}
$$

By an exchange of dummy indices, we can also write Eq. (26) as

$$
\boldsymbol{J} = -\sum_i\sum_{j\neq i}\boldsymbol{r}_{ij}\left(\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}}\cdot \boldsymbol{v}_j\right). \tag{27}
$$

Both Eqs. (26) and (27) can be used in the Green–Kubo method for thermal conductivity calculations, but Eq. (26) is a more convenient form for the homogeneous nonequilibrium molecular dynamics (HNEMD) method and the related spectral decomposition method, as it does not involve the velocities $\boldsymbol{v}_j$ of the neighboring atoms $j$ for a given atom $i$.

The heat current expressions above apply to all the interatomic potentials implemented in GPUMD. In all the cases, the validity of the heat current expressions has been numerically confirmed in terms of energy conservation.[62–65] Equation (27) has been recently used in an on-the-fly MLP.[66] Since the only assumptions for the derivations[60] of these expressions are the locality properties of the interatomic potentials, our heat current expressions generally apply to the multi-body cluster potentials as considered in Ref. 67, as we show in Appendix A.

## F. Loss function and training algorithm

We use the separable natural evolution strategy (SNES)[68] to optimize the free parameters in NEP. We denote a set of parameters


Published under an exclusive license by AIP Publishing

as a vector $\mathbf{z}$, whose dimension is the total number of parameters $N_{\rm par}$. In NEP1,

$$N_{\rm par} = (N_{\rm des} + 2)N_{\rm neu} + 1, \tag{28}$$

which is the same for both single-component and multicomponent systems. In NEP2, we added trainable parameters to the descriptor for multicomponent systems, and we have

$$N_{\rm par} = (N_{\rm des} + 2)N_{\rm neu} + 1 + N_{\rm typ}^2(n_{\rm max}^{\rm R} + n_{\rm max}^{\rm A} + 2) \tag{29}$$

if $N_{\rm typ} > 1$ and the same $N_{\rm par}$ as in NEP1 if $N_{\rm typ} = 1$. In NEP3, the number of trainable parameters in the descriptor is increased for both single and multicomponent systems and we have

$$\begin{aligned} N_{\rm par} = {} & (N_{\rm des} + 2)N_{\rm neu} + 1 + N_{\rm typ}^2(n_{\rm max}^{\rm R} + 1)(N_{\rm bas}^{\rm R} + 1) \\ & + N_{\rm typ}^2(n_{\rm max}^{\rm A} + 1)(N_{\rm bas}^{\rm A} + 1). \end{aligned} \tag{30}$$

We can formally express the loss function as a function of the free parameters,

$$L = L(\mathbf{z}), \tag{31}$$

and express the training process as a real-valued optimization problem,

$$\mathbf{z}^* = \arg\min_{\mathbf{z}} L(\mathbf{z}), \tag{32}$$

where $\mathbf{z}^*$ is an optimal set of parameters.

The total loss function is defined as a weighted sum of all these individual loss functions,

$$\begin{aligned} L(\mathbf{z}) = {} & \lambda_{\rm e}\left(\frac{1}{N_{\rm str}}\sum_{n=1}^{N_{\rm str}}\left(U^{\rm NEP}(n,\mathbf{z}) - U^{\rm tar}(n)\right)^2\right)^{1/2} \\ & + \lambda_{\rm f}\left(\frac{1}{3N}\sum_{i=1}^{N}\left(\boldsymbol{F}_i^{\rm NEP}(\mathbf{z}) - \boldsymbol{F}_i^{\rm tar}\right)^2\right)^{1/2} \\ & + \lambda_{\rm v}\left(\frac{1}{6N_{\rm str}}\sum_{n=1}^{N_{\rm str}}\sum_{\mu\nu}\left(W_{\mu\nu}^{\rm NEP}(n,\mathbf{z}) - W_{\mu\nu}^{\rm tar}(n)\right)^2\right)^{1/2} \\ & + \lambda_1\frac{1}{N_{\rm par}}\sum_{n=1}^{N_{\rm par}}|z_n| + \lambda_2\left(\frac{1}{N_{\rm par}}\sum_{n=1}^{N_{\rm par}}z_n^2\right)^{1/2}, \end{aligned} \tag{33}$$

where $N_{\rm str}$ is the number of structures in the training dataset (if using a full batch) or the number of structures in a mini-batch and $N$ is the total number of atoms in these structures. $U^{\rm NEP}(n,\mathbf{z})$ and $W_{\mu\nu}^{\rm NEP}(n,\mathbf{z})$ are per-atom energy and virial tensor predicted by the NEP with parameters $\mathbf{z}$ for the $n$th structure, and $\boldsymbol{F}_i^{\rm NEP}(\mathbf{z})$ is the predicted force for the $i$th atom. $U^{\rm tar}(n)$, $W_{\mu\nu}^{\rm tar}(n)$, and $\boldsymbol{F}_i^{\rm tar}$ are the corresponding target values. That is, the loss functions for energy, force, and virial are defined as their root mean square errors (RMSEs) between the current NEP predictions and the target values. The last two terms represent $\ell_1$ and $\ell_2$ regularization. The weights $\lambda_{\rm e}$, $\lambda_{\rm f}$, $\lambda_{\rm v}$, $\lambda_1$, and $\lambda_2$ are tunable hyperparameters. When calculating the loss function, we use the following units: eV/atom for energy and virial and eV/Å for force components.

The SNES[68] we use for optimizing Eq. (33) is a principled approach to real-valued evolutionary optimization by following the natural gradient of the loss function to update a search distribution (a mean value and a variance for each trainable parameter) for a population of solutions. It is a derivative-free black-box optimizer and, thus, does not require the loss function to have any analytical property. An explicit workflow of the training algorithm has been presented in Ref. 41.

### G. GPU implementation

The NEP approach is implemented in the open-source GPUMD package, which is a general-purpose molecular dynamics (MD) simulation package fully implemented on GPUs. It currently supports only Nvidia GPUs and the programming language is CUDA C++. In this section as well as Appendix B, we present the detailed algorithms for our CUDA implementation of the NEP approach.

Similar to many other MLPs, NEP is a many-body potential and is very similar to empirical many-body potentials such as the EAM potential[40] and the Tersoff potential.[38] Specifically, the radial descriptor part resembles an EAM potential and the angular descriptor part resembles a Tersoff potential. Therefore, our CUDA implementation of the NEP approach follows the established efficient scheme for empirical many-body potentials.[44]

The overall strategy of our CUDA implementation of the NEP approach is to use a few CUDA kernels only, which ensures a high degree of parallelism and high arithmetic intensity, both of which are crucial for achieving high performance in CUDA programming. In all the CUDA kernels, one CUDA thread is assigned to one atom, i.e., there is a one-to-one correspondence between atoms and CUDA threads. The descriptor vector and the various per-atom quantities, including the site energy, force, and virial for one atom, are all calculated within one CUDA thread. This also includes the application of the neural network as represented by Eq. (1). Appendix B1 shows the CUDA __device__ function (to be called in the first CUDA kernel function as discussed below) evaluating Eq. (1) as well as $\{\partial U_i/\partial q_v^i\}$. One can see that the feedforward neural network used here is an analytical multivariable scalar function.

The evaluation of NEP related quantities (energy, force, and virial) requires four CUDA kernels only:

1. The first CUDA kernel calculates the whole descriptor vector and applies the neural network to obtain the site energy and the derivatives of the energy with respect to the descriptor components, see Algorithm 1 in Appendix B.
2. The second CUDA kernel calculates the force and virial related to the radial descriptor components, see Algorithm 2 in Appendix B.
3. The third CUDA kernel calculates the partial force related to the angular descriptor components, see Algorithm 3 in Appendix B.
4. The fourth CUDA kernel calculates the force and virial related to the angular descriptor components, see Algorithm 4 in Appendix B.

In these CUDA kernels, the inputs and outputs related to the atoms (position, energy, force, and virial) are all in double precision, but the internal calculations within the CUDA kernels are mostly in single precision. This is an effective mixed-precision approach


Published under an exclusive license by AIP Publishing

capable of maintaining a good balance between accuracy and efficiency that has been adopted in many other GPU-accelerated atomistic simulation packages.[69–71]

## H. Training and using NEPs

### 1. Overview of the GPUMD package

The GPUMD package[43,44] can be used to train NEPs and use them in atomistic simulations. GPUMD can be compiled and used in both Linux and Windows, provided that a CUDA development environment and a CUDA-enabled GPU are available. After compilation, one obtains the nep and gpumd executable, which can be used to train and use NEPs, respectively. Figure 1 provides a schematic overview of the workflow of training and using NEPs.

The GPUMD package started from a minimal CUDA code implementing only simple pairwise potentials and thermal conductivity calculations.[43] Gradually, empirical many-body potentials were implemented using the unique formalism we proposed,[44,60] including the Tersoff [38] and Tersoff-like[72] potentials, the Stillinger–Weber potential,[73] and EAM potentials.[40] Recently, support was also added for machine-learned force constant potentials constructed using the HIPHIVE package.[74,65] The most recent additions are the various versions of NEP as developed in the previous papers[41,42] as well as the current one.

Apart from supporting the above important interatomic potentials, GPUMD also supports many statistical ensembles, including the NVE (microcanonical), NVT (isothermal), and NPT (isothermal–isobaric) ensembles. For the NVT ensemble, it has options for the Berendsen thermostat,[75] the Nosé–Hoover chain thermostat,[76–78] the Bussi–Donadio–Parrinello thermostat,[79] and the Langevin thermostat in different flavors.[80,81] For the NPT ensemble, GPUMD supports the classical Berendsen barostat[75] and the recently proposed stochastic cell-rescaling barostat by Bernetti and Bussi.[82]

GPUMD has been mainly used to study thermal transport. It supports the equilibrium MD method based on the Green–Kubo relation,[83,84] the nonequilibrium MD method using two or more thermostats (preferably the Langevin thermostats[85]), and the HNEMD method using a homogeneous driving force.[86,87] The thermal transport coefficients, either thermal conductance or thermal conductivity, can be decomposed both spatially and spectrally.[61,63,87] GPUMD has been used to establish the best practices[85,88] in MD simulations of thermal transport. It has also been used to study the particular thermal transport properties of specific materials, including various two-dimensional (2D) materials,[64,89–103] vdW structures based on 2D materials,[104–110] and quasi-one-dimensional materials.[111–113] There are applications focused on revealing unique phonon transport mechanisms.[114–121] The high efficiency of GPUMD

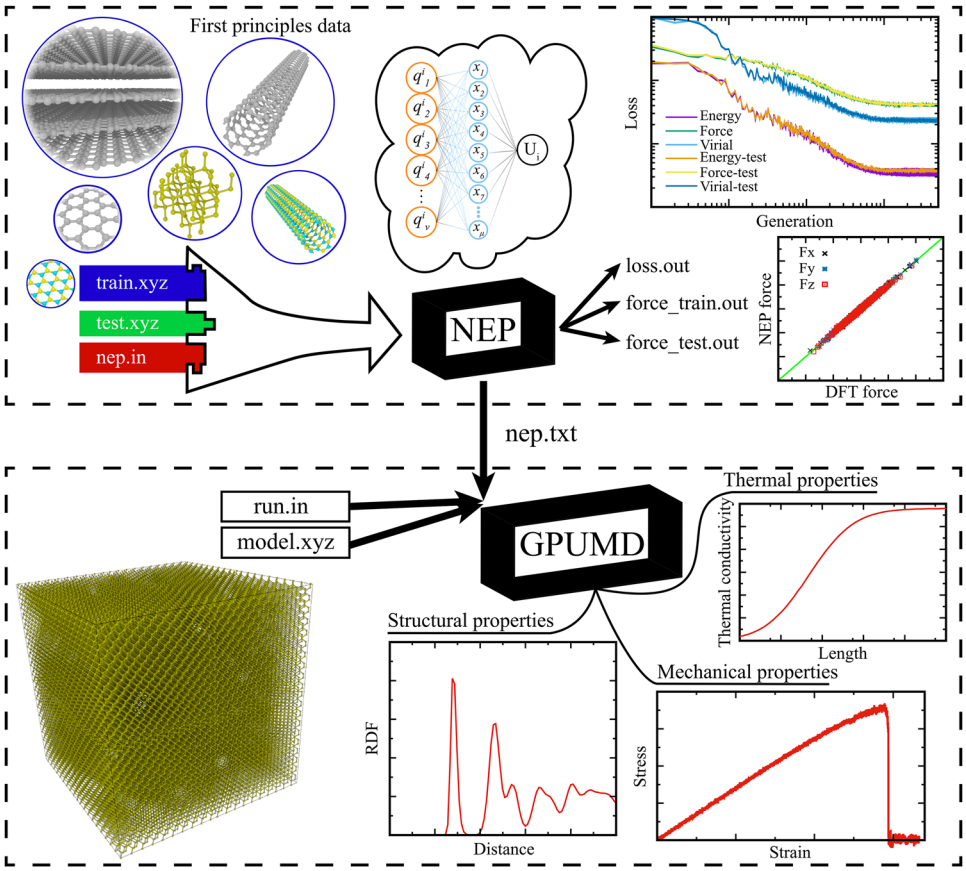


**FIG. 1.** The GPUMD package includes two executables, viz., `gpumd` and `nep`, which are represented by the two black boxes. The `nep` program can be used for training NEP models and the `gpumd` program can be used to perform atomistic simulations using the trained potentials. See text for details.


Published under an exclusive license by AIP Publishing

also enabled high-throughput thermal transport simulations that were used as training/testing data for machine learning models of interfacial thermal transport.[122]

Although previous studies using GPUMD have been mostly focused on thermal transport properties, GPUMD has already been developed into a general-purpose atomistic simulation package. In Sec. V, we will showcase a series of typical atomistic simulations using a NEP model for carbon systems trained in this paper.

### 2. Training a NEP model

To train a NEP model, one needs to prepare three input files: train.xyz, test.xyz, and nep.in. The first two contain the training and the testing data, respectively. The files train.xyz and test.xyz have the same data format, the only difference being that the data in train.xyz will be used for training and those in test.xyz will be used for testing. The file train.xyz (test.xyz) contains the following data: the number of structures in the training (testing) set, the reference energy, reference virial tensor (optional), and cell metric for each structure, as well as the chemical symbol, position vector, and reference force vector for each atom in each structure. For the specific data format, we refer the reader to the GPUMD manual.

The file nep.in contains hyperparameters that define the NEP model and control the training process. In this file, one can choose the NEP version (currently NEP2 or NEP3), specify the number of atom types and their chemical symbols, the cutoff distances $r_c^R$ and $r_c^A$, the radial function parameters ($n_{max}^R$, $n_{max}^A$, $N_{bas}^R$, and $N_{bas}^A$), the angular expansion parameters ($l_{max}^{3b}$, $l_{max}^{4b}$, and $l_{max}^{5b}$), the weights for the different terms in the loss function ($\lambda_e$, $\lambda_f$, $\lambda_v$, $\lambda_1$, and $\lambda_2$), the number of neurons $N_{neu}$ in the hidden layer of the neural network, the batch size $N_{bat}$ (number of structures in one batch), the population size $N_{pop}$, and the number of generations $N_{gen}$ in the SNES training algorithm. Details concerning the nep.in file are presented in the GPUMD manual.

During the training process, predicted energy, force, virial values, the various terms of the loss function (both for the training and the testing datasets), the potential file, and a file used for restarting are continuously updated. Further details concerning the output files of the nep executable can be found in the GPUMD manual.

### 3. Using a NEP

The potential file nep.txt contains all the information that constitutes a NEP model and can be used directly as an input to the gpumd executable for running atomistic simulations. To use the gpumd executable, one needs to prepare another input file, run.in, which specifies the simulation process. Several examples are presented in Sec. V below.

## III. PERFORMANCE OF NEP MODELS

### A. NEP3 vs NEP2

In this section, we demonstrate the workflow of using the nep executable to train NEPs and show the enhanced accuracy of NEP3 compared to NEP2 due to the improved radial functions. Here, we use the MgAlCu alloy system, which has been studied previously using a DP.[123] There are 141 409 structures and we used 90% of this set for training and 10% for testing. There are tools in the GPUMD package for preparing the required `train.xyz` and `test.xyz` input files for the `nep` executable.

The other input file needed for the nep executable is nep.in. For our test using NEP2, this file reads as follows:

```
version     2
type        3 Cu Al Mg
cutoff      6 3.5
n_max       15 10
l_max       4
neuron      50
lambda_1    0.05
lambda_2    0.05
lambda_e    1.0
lambda_f    1.0
lambda_v    0.1
batch       1000
population  50
generation  500000
```

For our test using NEP3, it reads as follows:

```
version     3
type        3 Cu Al Mg
cutoff      6 3.5
n_max       12 8
basis_size  12 8
l_max       4
neuron      50
lambda_1    0.05
lambda_2    0.05
lambda_e    1.0
lambda_f    1.0
lambda_v    0.1
batch       1000
population  50
generation  500000
```

The only differences between the hyperparameters used for NEP2 and NEP3 are related to the radial functions. For NEP2, we use $n_{max}^R = 15$ and $n_{max}^A = 10$. For NEP3, we use $n_{max}^R = N_{bas}^R = 12$ and $n_{max}^A = N_{bas}^A = 8$. The reason for using smaller $n_{max}^R$ and $n_{max}^A$ values in NEP3 compared to NEP2 is that one radial function in NEP3 is a linear combination of a few basis functions, while one radial function in NEP2 is simply one basis function. Even in this case, NEP3 can achieve a noticeably higher accuracy than NEP2, as shown in Table I. With these parameters, NEP3 still has a computational

**TABLE I.** RMSEs for energies, forces, and virials for the MgAlCu alloy system using NEP2 and NEP3.

| RMSE | NEP2 | NEP3 |
|---|---|---|
| Energies (meV/atom) | 11 | 10 |
| Forces (meV/Å) | 68 | 63 |
| Virials (meV/atom) | 43 | 41 |


Published under an exclusive license by AIP Publishing

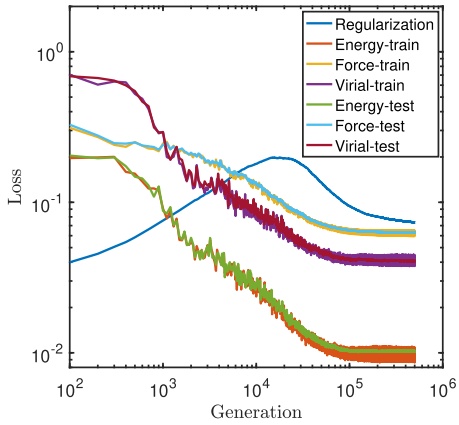


**FIG. 2.** Evolution of the various terms in the loss function (33) with respect to the generation in the SNES[68] training algorithm for the MgAlCu training and test datasets.[123]

speed similar to that of NEP2 in MD simulations, reaching about $1.5 \times 10^7$ atom step/s using one A100 GPU.

The `nep` executable produces a number of output files that can be used to examine the training/testing results in detail. Figure 2 shows the training and testing RMSEs as well as the loss function related to the regularization (from the loss.out file). The training RMSEs exhibits oscillations because of the use of mini-batches (with a batch size of 1000). The test RMSEs closely follow the training RMSEs, which indicates the very good interpolation capability of NEP. In other words, there is no sign of over-fitting. As shown in Ref. 41, a proper regularization is crucial to prevent possible over-fitting in NEP models. Figure 3 shows the results from the force_test.out file. We can see that both NEP2 and NEP3 achieve a rather high level of accuracy here.

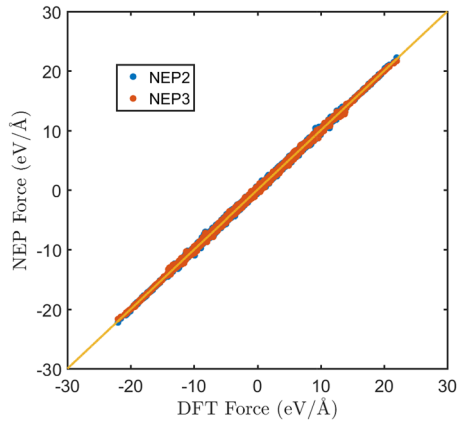


**FIG. 3.** Forces from NEP2 and NEP3 models against the target DFT values for the MgAlCu test set. The solid line represents the identity function that serves as a guide to the eye.

## B. Comparison of NEP3 with other MLPs

### *1. A general-purpose silicon dataset*

We use the general-purpose silicon training dataset from Ref. 124 to test convergence with respect to some hyperparameters and compare the results from an implementation of the ACE.[48] This dataset consists of 2475 structures, including bulk crystal structures, $sp^2$ bonded structures, dimers, liquid structures, amorphous structures, diamond structures with surfaces or vacancies, and several other defective structures. Every structure has an energy, but not all the structures have virial data. For details on the reference density functional theory (DFT) calculations, the reader is referred to Ref. 124.

We use the same cutoff distance of 5 Å for the radial and angular parts and set $n^{\mathrm{R}}_{\mathrm{max}} = n^{\mathrm{A}}_{\mathrm{max}} = N^{\mathrm{R}}_{\mathrm{bas}} = N^{\mathrm{A}}_{\mathrm{bas}} = 10$. We consider using three-body descriptor components only ($l^{\mathrm{3b}}_{\mathrm{max}} = 4$, $l^{\mathrm{4b}}_{\mathrm{max}} = 0$, $l^{\mathrm{5b}}_{\mathrm{max}} = 0$), using both three-body and four-body descriptor components ($l^{\mathrm{3b}}_{\mathrm{max}} = 4$, $l^{\mathrm{4b}}_{\mathrm{max}} = 2$, $l^{\mathrm{5b}}_{\mathrm{max}} = 0$), and using up to five-body descriptor components ($l^{\mathrm{3b}}_{\mathrm{max}} = 4$, $l^{\mathrm{4b}}_{\mathrm{max}} = 2$, $l^{\mathrm{5b}}_{\mathrm{max}} = 1$). Other common hyperparameters are as follows: $\lambda_1 = \lambda_2 = 0.05$, $\lambda_{\mathrm{e}} = 1$, $\lambda_{\mathrm{f}} = 1$, $\lambda_{\mathrm{v}} = 0.1$, $N_{\mathrm{neu}} = 50$, $N_{\mathrm{bat}} = \mathrm{full}$, $N_{\mathrm{pop}} = 50$, and $N_{\mathrm{gen}} = 3 \times 10^5$.

Figure 4 shows the force RMSE vs the computational cost of force evaluation for NEP models and previous results for an implementation[48] of the ACE potential[46] based on linear regression. While the ACE potential shows a strong dependence of the accuracy on the maximum correlation order of the angular descriptor components, our NEP shows a much weaker dependence, although considering four-body and five-body correlations indeed helps to increase the accuracy to some degree. With three-body descriptor components only, the ACE potential has a force RMSE above 1 eV/Å,

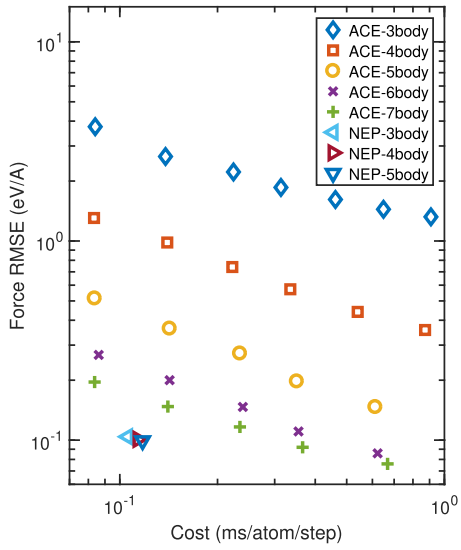


**FIG. 4.** Force RMSE against computational cost for NEP models for silicon compared to an implementation of the ACE approach (the faster recursive approach as in Ref. 48). A serial C++ implementation of the NEP approach has been tested using an Intel i7-8750H CPU @ 2.2 GHz. For comparison, the ACE potential was implemented in Julia and tested using an Intel i7-7820HQ CPU @ 2.9 GHz.[48]


Published under an exclusive license by AIP Publishing

while the three-body NEP model already achieves an accuracy of about 0.1 eV/Å. To achieve the same accuracy as the NEP models, one needs to consider up to six-body descriptor components in the ACE approach. The reason for the relative faster convergence of the accuracy with respect to the correlation order for the NEP models compared to the ACE model is probably due to the use of a neural network as the regression method instead of linear regression as used in the ACE approach. Using linear regression, the descriptor needs to be rather complete, which can easily lead to more than $10^4$ descriptor components,[48] while the descriptor vector lengths range from 55 to 77 in the present NEP models. With a reduced descriptor length, the completeness[48] of the descriptor will be reduced to some degree, but the incompleteness of the descriptor can be (partially) compensated by the neural network and the overall computational cost at a given target accuracy can be lower than using a large number of descriptor components and linear regression. Indeed, within the framework of $N$-body iterative contraction of equivariants (NICE), nonlinear neural network regression has been shown to be able to achieve higher accuracy than linear regression at least in the limit of large training set size,[125] using a descriptor up to the same level of the $N$-body correlation.

### 2. Azobenzene molecule

Our next example is the largest molecule, azobenzene, in the MD17 dataset.[126] This dataset has been revised later[127] to ensure more strict convergence in the DFT calculations, which is referred to as the revised MD17 dataset (rMD17). Here, we use the first train–test split as reported in rMD17,[127] with 1000 training structures and 1000 testing structures randomly chosen from a MD trajectory at 500 K. There are, thus, 1000 target energies and 72 000 target force components in the training dataset but there are no target virials.

As we have confirmed that adding four-body and five-body descriptor components can lead to higher accuracy, here we use $l^{3b}_{max} = 4$, $l^{4b}_{max} = 2$, $l^{5b}_{max} = 1$ and consider different values of the radial function hyperparameters. For simplicity, we set $n^R_{max} = N^R_{bas}$ and $n^A_{max} = N^A_{bas}$ and consider the following combinations of parameters: $(n^R_{max}, n^A_{max}) = (6,4)$, $(9,6)$, $(12,8)$, and $(15,10)$. The other hyperparameters are $r^R_c = 6$ Å, $r^A_c = 4$ Å, $\lambda_1 = \lambda_2 = 0.02$, $\lambda_e = 1$, $\lambda_f = 1$, $\lambda_v = 0$, $N_{neu} = 50$, $N_{bat} =$ full, $N_{pop} = 50$, and $N_{gen} = 10^6$.

Figure 5 shows the force mean absolute error (MAE) vs the computational cost of force evaluation for our NEP and some other MLPs as reported in Ref. 47, including ANI,[15] GAP,[8] sGDML,[24] and a linear-regression based ACE potential[47] (similar to but not identical to the one in Ref. 48). With increasing $n^R_{max}$ and $n^A_{max}$, both the accuracy and computational cost increase quickly. With $n^R_{max} = 15$ and $n^A_{max} = 10$, the NEP approach achieves an accuracy between ANI and GAP, but it is more than one order of magnitude faster than ANI and more than two orders of magnitude faster than GAP. At this level of accuracy, the NEP models are also faster than the linear-ACE potentials, while the linear-ACE potentials can reach higher accuracy with further increased cost. Similar to the case of silicon, we propose that the superior cost effectiveness of the NEP models as compared to the linear-ACE potentials is due to the much smaller descriptor vector size in NEP, which ranges from 32 to 82 in the NEP models, but from 1700 to 122 000 in the linear-ACE potentials for the test cases in Fig. 5.

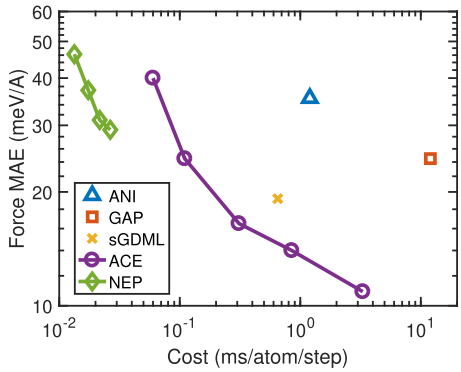


**FIG. 5.** Force MAE against computational cost from NEP models and several other MLPs for azobenzene as reported in Ref. 47. To be consistent with Ref. 47, we tested our serial C++ code using an Intel Xeon Gold 5218 @ 2.3 GHz.

### 3. Carbon dataset

In this section, we use the training and test datasets for the carbon system with various phases[10] to compare the NEP approach with other MLPs in terms of accuracy, speed, and memory usage. The MLPs to be compared include DP,[17] GAP,[8] MTP,[30] and REANN.[45] The training dataset comprises 4080 structures in total, including bulk crystal structures, bulk liquid and amorphous structures, amorphous surfaces, and isolated dimer structures. The testing dataset comprises 450 structures similar to those in the training dataset, but excluding the dimer ones. There are 256 628 and 28 337 atoms in the training and testing datasets, respectively. Each structure has one target energy and each atom has three target force components. Some structures also have target virials. For more details on the datasets, see Ref. 10.

In the case of the NEP approach, we used the NEP3 form and considered two sets of hyperparameters. In the first one, we set $r^R_c = 4.2$ Å, $r^A_c = 3.7$ Å, $n^R_{max} = N^R_{bas} = 10$, $n^A_{max} = N^A_{bas} = 8$, $l^{3b}_{max} = 4$, $l^{4b}_{max} = 2$, $l^{5b}_{max} = 1$, $\lambda_1 = \lambda_2 = 0.05$, $\lambda_e = 1$, $\lambda_f = 1$, $\lambda_v = 0.1$, $N_{neu} = 100$, $N_{bat} =$ full, $N_{pop} = 50$, and $N_{gen} = 5 \times 10^5$. This model is labelled "NEP (4.2 Å)" in Table II. In the second one, we make the following changes as compared to the first one: $r^R_c = 3.7$ Å, $r^A_c = 3.2$ Å, $l^{5b}_{max} = 0$, and $N_{neu} = 50$. This model is labelled "NEP (3.7 Å)" in Table II.

For DP, we used the DEEPMD-KIT package[17] and the smooth edition.[19] We trained two versions of DP: one using the se_a descriptor (with a cutoff of 6 Å) only and the other using a combination of `se_e2_a` (with a cutoff of 6 Å) and `se_e3` (with a cutoff of 3.8 Å). These two versions are labeled "DP (se2)" and "DP(se2+se3)" in Table II, respectively. The size of the embedding net is $(25, 50, 100)$ for the `se_a` and `se_e2_a` descriptors and $(20, 40, 80)$ for the `se_e3` descriptor, and the size of the fitting net is $(240, 240, 240)$. The learning rate decreases exponentially from $10^{-3}$ to $10^{-8}$. The weighting parameters for energy, force, and virial have a starting value of 0.02, 1000, and 0.01, respectively, which are linearly changed to 1, 1, and 0.1 during the training process. The number of training steps is $10^7$, which is sufficiently large.

For GAP,[8,10] we directly took the results from Ref. 10. The $n_{max}$ and $l_{max}$ for the smooth overlap of atomic positions (SOAP)


Published under an exclusive license by AIP Publishing

**TABLE II.** Performance comparison between NEP models and other MLPs for the carbon test set from Ref. 10. Energy RMSE $\Delta E$ and virial RMSE $\Delta W$ are in units of meV/atom while the force RMSE $\Delta F$ is in units of meV/Å. Computational speed is measured in atom step/ms. $N_{max}$ is the maximum number of atoms that can be simulated using one V100 GPU for the three GPU-accelerated codes. For GAP[8,10] and MTP,[30,31] 72 Intel Xeon-Gold 6240 CPU cores were used. For DP[17] after compression, NEP and REANN,[32,45] a 32 GB V100 GPU was used.

| MLP | $\Delta E$ | $\Delta F$ | $\Delta W$ | Speed | $N_{max}$ |
|---|---|---|---|---|---|
| GAP | 46 | 1100 | NA | 6.1 | NA |
| DP (se2) | 80 | 1100 | 250 | 290 | $240 \times 10^3$ |
| DP (se2+se3) | 44 | 800 | 170 | 150 | $220 \times 10^3$ |
| MTP (4 Å) | 36 | 650 | 180 | 110 | NA |
| MTP (5 Å) | 35 | 630 | 200 | 61 | NA |
| MTP (6 Å) | 35 | 650 | 220 | 27 | NA |
| NEP (4.2 Å) | 42 | 690 | 160 | 3600 | $10\,000 \times 10^3$ |
| NEP (3.7 Å) | 44 | 700 | 170 | 4600 | $12\,000 \times 10^3$ |
| REANN (3 Å) | 41 | 700 | NA | 280 | $290 \times 10^3$ |
| REANN (4 Å) | 31 | 640 | NA | 170 | $180 \times 10^3$ |
| REANN (6 Å) | 28 | 670 | NA | 62 | $64 \times 10^3$ |

descriptor were both set to 8.[10] There were also separate low-dimensional two-body and three-body components in this GAP.[10]

For MTP,[30] we used the MLIP package.[31] The descriptor level of the MTP is set to 22. We considered three cutoff distances, viz., 4, 5, and 6 Å, labeled "MTP (4 Å)," "MTP (5 Å)," and "MTP (6 Å)," respectively, in Table II.

For REANN,[45] we used the REANN package.[32] We considered three cutoff distances, viz., 3, 4, and 6 Å, labeled "REANN (3 Å)," "REANN (4 Å)," and "REANN (6 Å)," respectively, in Table II. The weighting parameter for energy is kept at 1 and that for force is decreased from 10 to 0.5 during the training process. A batch size of 32 is used (we have tried to use a larger batch size and it turned out to exceed the memory limit of a 32 GB V100). The sizes of the neural networks for both the atom energy and the orbital coefficients are $(64, 64)$. The learning rate decreases exponentially from $10^{-3}$ to $10^{-7}$. The number of training epochs is $10^4$.

For all MLPs, we list the RMSEs for energies, forces, and virials (calculated from six independent components) in Table II. The MTP and REANN models show the best accuracy in energies and forces, and GAP and DP the worst. The accuracy of the NEP models is close to those of MTP and REANN. The virial RMSE is missing for GAP as no predicted virial data have been provided.[10] It is also missing for REANN because the virial has not been formulated in this MLP. For the MLPs with available virial data, the NEP models achieve the highest accuracy. Therefore, we can say that the NEP models at least have an above-average accuracy for this carbon dataset.

With the accuracy comparison results in mind, we next compare the computational performance in realistic atomistic simulations. Here, we run MD simulations for a cubic cell of diamond in the isothermal ensemble at 300 K for 100 steps and output some basic thermodynamic properties every ten steps. Based on the MD simulations, we measure the computational speed as the product of the number of atoms and the number of steps divided by the total wall time used. Three of the MLPs (DP, REANN, NEP) have been implemented on GPUs and we, thus, use an Nvidia V100 GPU for the test. For the other two MLPs for which there are only CPU versions available (GAP and MTP), we use 72 Intel Xeon-Gold 6240 cores. The CPU and GPU resources might have unequal financial costs, but one can make suitable conversions of the results presented here to other computational environments. For the CPU-based MLPs, MTP shows much higher computational speed than GAP, which is consistent with previous tests.[30] For the GPU-based MLPs, DP (after model compression) and REANN have comparable speed, while the NEP models are more than one order of magnitude faster. Figure 6 shows that the NEP models substantially lower the Pareto front of accuracy-versus-cost that can be achieved by the other MLPs.

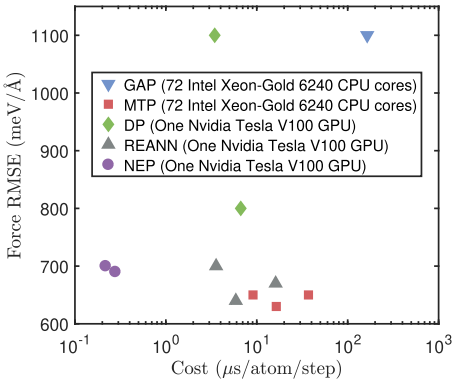


**FIG. 6.** Force RMSE against computational cost from NEP models and other MLPs for the carbon dataset from Ref. 10.

Interestingly, the GPU memory usage seems to be correlated to the computational speed: The maximum number of atoms $N_{max}$ that can be simulated using one V100 GPU is roughly proportional to the computational speed. This comparison highlights the superior computational performance of the NEP approach as implemented in GPUMD in terms of both computational speed and memory efficiency, which is crucial for tackling challenging applications that require large-scale and long-time atomistic simulations.

## IV. ACTIVE LEARNING BASED ON THE LATENT SPACE

Apart from regression accuracy and MD speed, training data preparation is an important aspect of MLPs. Because quantum-mechanical calculations are usually time-consuming, it is desirable to construct a minimal training dataset for a given application. One strategy for achieving this is to use active learning (AL), which uses query criteria to determine whether or not a new training sample should be included into an existing training set to improve the model accuracy and generalization capability. Many AL schemes have been proposed for MLPs, including the ensemble (or query-by-committee) method,[128–130] the dropout method,[131] methods based on feature-space distance measuring[132] and entropy maximization in the descriptor space,[133] and methods based on


Published under an exclusive license by AIP Publishing

optimal design.[134,135] Here, we propose an AL scheme based on the latent space of a pre-trained NEP model. This AL scheme has been inspired by the work of Janet *et al.*,[136] who have shown that distance in the latent space provides a good quantitative uncertainty metric to be used in an AL scheme.

There is no unique definition of the latent space. Here, we define it as an $N_{\rm neu}$-dimensional space spanned by the vectors whose components are the product of the states of the hidden-layer neurons and the connection weights between them and the output layer. To compute the latent space vector for a structure, one must train a NEP model first, but this can be achieved by using a small initial training dataset. Then, one can use the pre-trained NEP model to compute the latent space vectors for many structures, either those in the training dataset or new ones that have no target values (energy, forces, and virials) yet. This allows one to generate target values (via quantum-mechanical calculations) for a number of structures that have relatively large distances to existing points in the latent space. This procedure can be iterated by updating the training dataset and the NEP model in alternating fashion. During this process, the existing NEP model can be used to create the new structures to be examined, using various sampling techniques in atomistic simulations.

We take the carbon dataset as a concrete example to illustrate the idea outlined above. To this end, suppose we only have 200 structures randomly selected out of the 4080 ones in the original training dataset. We first train an initial NEP using these 200 structures, adopting the same hyperparameters as used for the NEP (4.2 Å) model in Table II. Assuming that we have obtained the remaining structures as in the original training dataset by various means, we then compute the latent space vectors for all the 4080 structures. One can define a distance in the high-dimensional latent space, but it turns out that the high dimension can be effectively reduced using principal component (PC) analysis. The explained variance ratios of the first 30 PCs are shown in Fig. 7. The first two leading PCs contribute more than 70% to the total dimensions, allowing us to visualize the distribution of structures in a 2D PC space, as shown in Fig. 8. It can be seen that both dimers and crystals (with or without defects) occupy a small area in the PC space. On the other hand, both bulk and surface amorphous/liquid structures occupy large areas that are almost overlapping. Furthermore, the high density part in Fig. 8(a) indicates that there are relatively more bulk amorphous and liquid structures in the full training dataset.

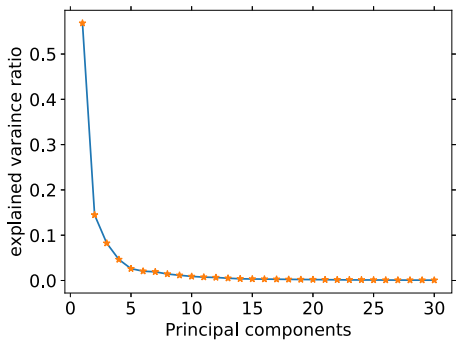


**FIG. 7.** Normalized explained variance ratio of the first 30 principal components calculated from the 4080 structures in the original training dataset[10] based on the initial NEP model trained using 200 structures.

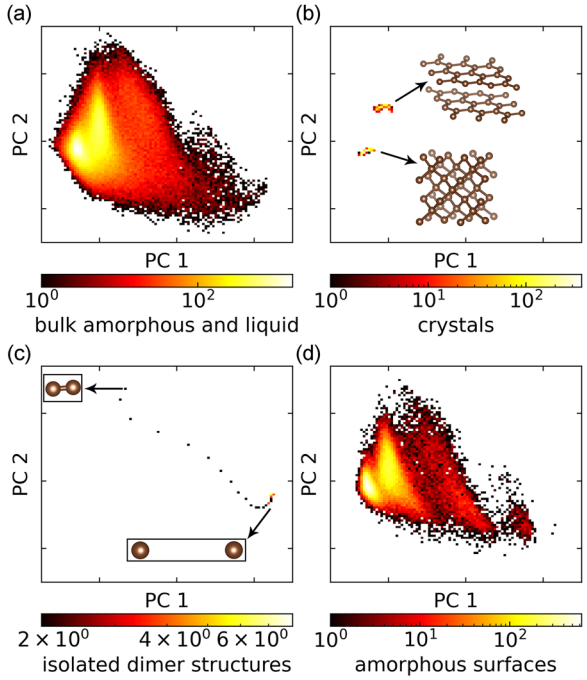


**FIG. 8.** Distribution of the 4080 carbon structures in the full training dataset[10] in the 2D principal component (PC) space (spanned by PC 1 and PC 2) as reduced from the latent space that was constructed using the initial NEP model trained using 200 structures. (a) Bulk amorphous/liquid structures, (b) crystals including $sp^2$ graphite and $sp^3$ diamond structures, (c) dimers, and (d) surface amorphous structures. The color bar represents the density of structures in the 2D PC space.

Based on these observations, we construct a new training dataset that includes all the dimers (30 in total) and crystals (356 in total), and 400 bulk amorphous/liquid structures obtained by farthest-point sampling. Using these 786 structures, we train a new NEP model using the same hyperparameters as for the NEP (4.2 Å) model in Sec. III B 3. The energy, force, and virial RMSEs from this NEP model are 45, 700, and 190 meV/atom, respectively, for the same test dataset as used in Sec. III B 3, which are very close to those for the NEP model trained using the full training dataset. Figure 9 shows that the AL-based NEP model indeed performs very well in the various predicted values. This is a notable result since we have not included a single surface amorphous structure into the training dataset for the AL-based NEP model. This shows that the distance in the latent space (and the reduced PC space) indeed provides a reliable metric for selecting new samples for the construction of accurate and transferable MLPs. The present results also indicate that the NEP approach is quite data efficient, which we attribute to the relatively simple neural network model and the inclusion of regularization terms in the loss function. As a further demonstration of the reliability of the AL-based NEP model, we show in Sec. V below that it performs equally well as the NEP model trained against


Published under an exclusive license by AIP Publishing

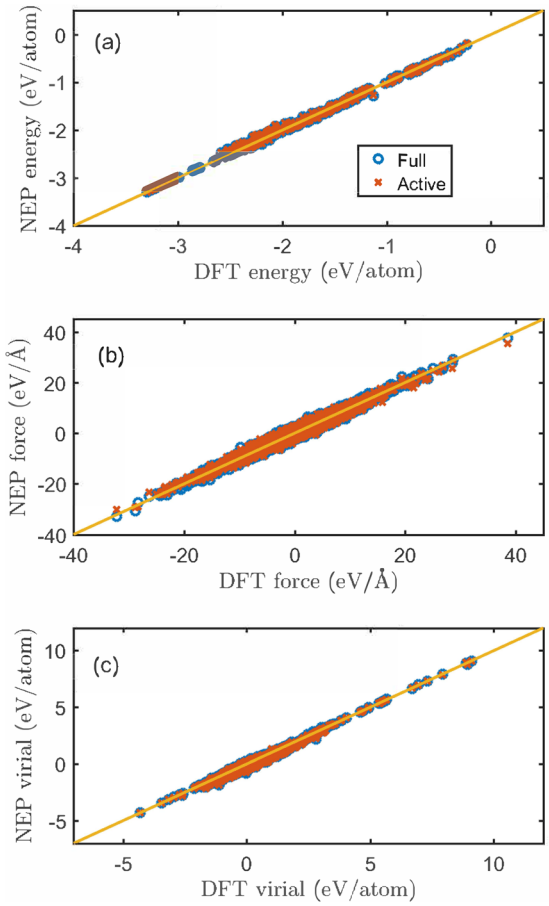


**FIG. 9.** (a) Energy, (b) force, and (c) virial values from the NEP models for carbon constructed using the full training dataset (4080 structures, labeled "Full") and the training dataset constructed based on active learning (786 structures, labeled "Active"), in comparison to the DFT reference data for the test dataset (450 structures).[10]

the full training dataset in an MD simulation covering a large range of temperatures.

## V. EXAMPLES FOR APPLICATIONS OF NEP MODELS

In this section, we demonstrate the application of NEP models in atomistic simulations. To this end, we employ the NEP (4.2 Å) model from Table II, if not stated otherwise.

### A. Lattice constant

We begin with a simple static calculation and determine the zero-temperature lattice constant of diamond by calculating a cohesive energy curve. The run.in input file reads as follows:

```
potential         C_2022_NEP3.txt
compute_cohesive  0.98 1.03 51
```

**TABLE III.** Structural and elastic properties of diamond from the NEP (4.2 Å) model in comparison to GAP and DFT-LDA results.[10]

| | DFT-LDA | GAP | NEP (4.2 Å) |
|---|---|---|---|
| $a$ (Å) | 3.532 | 3.539 | 3.530 |
| $C_{11}$ (GPa) | 1101 | 1090 | 1134 |
| $C_{12}$ (GPa) | 148 | 112 | 153 |
| $C_{44}$ (GPa) | 592 | 594 | 605 |

The `potential` keyword specifies the NEP model to be used and the `compure_cohesive` keyword is used to invoke the cohesive energy calculation. The lattice constant thus obtained is 3.530 Å, which is very close to the DFT reference value obtained using the local density approximation (LDA), see Table III.

### B. Elastic constants

Next, we compute the zero-temperature elastic constants, for which the run.in input file reads as follows:

```
potential         C_2022_NEP3.txt
compute_elastic   0.01 cubic
```

Here, the `compute_elastic` keyword is used to initiate the calculation of the three independent elastic constant components ($C_{11}$, $C_{12}$, and $C_{44}$) using the energy–strain relation with ±1% strain values. The computed elastic constants are presented in Table III. The elastic constants from the NEP model are within 4% of the reference DFT-LDA values. For comparison, the $C_{12}$ value from GAP is about 24% smaller than the reference DFT-LDA value. We note that including virial information during training is crucial for obtaining accurate elastic properties, as can be seen from Fig. 10. In other words, fitting to energy and force data alone does not guarantee an accurate description of virials. Since the calculation of the heat current involves virial terms, see Eq. (25), this is also important

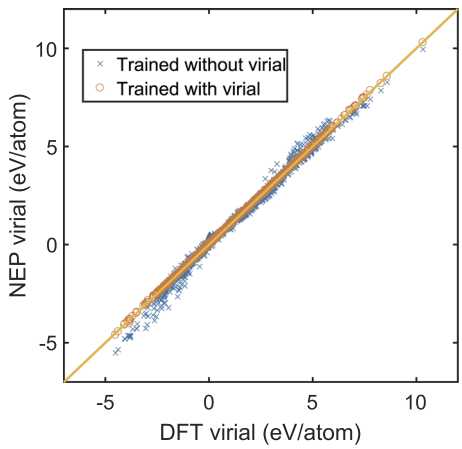


**FIG. 10.** Virial values from the NEP models in comparison with DFT-LDA data for the carbon dataset.[10]


Published under an exclusive license by AIP Publishing

for heat transport applications, as has already been pointed out by Shimamura *et al.*[137]

### C. Quenching

The carbon dataset[10] is particularly suitable for studying liquid and amorphous carbon.[10,138,139] In this example, we use a melt–quench–anneal protocol similar to that used in Ref. 10 (but with ten times longer simulation time for the relaxation at each temperature and an extra relaxation at 1000 K) to generate amorphous carbon. The `run.in` file reads as follows:

```
potential       C_2022_NEP3.txt
velocity        9000

ensemble        nvt_lan 9000 9000 100
time_step       1
dump_thermo     10
dump_position   1000
run             30000

ensemble        nvt_lan 5000 5000 100
dump_thermo     10
dump_position   1000
run             30000

ensemble        nvt_lan 5000 1000 100
dump_thermo     10
dump_position   100
run             500

ensemble        nvt_lan 1000 1000 100
dump_thermo     10
dump_position   1000
run             30000

ensemble        nvt_lan 300 300 100
dump_thermo     10
dump_position   1000
run             30000
```

The initial simulation model is a cubic diamond supercell containing 64 000 atoms with a mass density of 3.0 g cm$^{-3}$. The system is first quickly melted at 9000 K and then relaxed at 5000 K, followed by a quick quenching from 5000 to 1000 K and further relaxation at 1000 and 300 K. Here, the Langevin thermostat[80] is used to control the temperature. The evolution of temperature and the ratio of sp$^3$-bonded atoms as a function of simulation time are presented in Figs. 11(a) and 11(b). The radial and angular distribution functions $g(r)$ and $g(\theta)$ at 5000 and 300 K in Figs. 11(c) and 11(d) show that the system is in liquid and amorphous–solid states, respectively. We also performed the same MD simulation using the NEP model trained with the AL scheme in Sec. IV, and we can see that it gives almost identical results as those from the NEP model trained using the full training dataset. This further demonstrates the effectiveness of our AL scheme based on the latent space.

### D. Self-diffusion coefficient

To better distinguish between the liquid and amorphous states, we calculate the self-diffusion coefficient (SDC) for the carbon system at 5000 and 300 K.

The relevant part of the `run.in` file reads as follows:

```
ensemble        nve
time_step       0.5
compute_sdc     1 1000
run             10000
```

The SDC can be computed either from the mean square displacement based on the Einstein relation or from the VAC based on the Green–Kubo relation. We used the VAC approach here. The VAC is defined as

$$C(\tau) = \frac{1}{3N}\sum_i \langle \boldsymbol{v}_i(0)\cdot\boldsymbol{v}_i(\tau)\rangle, \tag{34}$$

and the running SDC is computed according to the following Green–Kubo relation:

$$D(t) = \int_0^t d\tau C(\tau). \tag{35}$$

The computation of the SDC is invoked by the `compute_sdc` keyword. The VAC and SDC are shown in Fig. 12. For both temperatures, the SDC shows a transition regime (as signified by the bump in the curves) within a short correlation time followed by a fully diffusive regime, which has a constant value. The system at 300 K has a zero SDC, which is expected for a solid state, while the system at 5000 K has a finite SDC of 1.334(3) Å$^2$/ps and is a liquid.

### E. Density of states and heat capacity of amorphous carbon

After obtaining amorphous carbon structures, we further study their thermal properties. In this example, we calculate the vibrational density of states (VDOS) for an amorphous carbon structure and then obtain the heat capacity with quantum corrections. The relevant part in the run.in file reads as follows:

```
ensemble    nve
time_step   1.0
compute_dos 5 200 400
run         100000
```

The VDOS $\rho(\omega)$ is calculated from the VAC[140] and is invoked by the compute_dos keyword. The VAC and VDOS are normalized to $3N$, where $N$ is the number of atoms. The per-atom heat capacity with quantum corrections at the temperature $T$ is then calculated as

$$C(T) = \frac{1}{N}\int_0^\infty \frac{d\omega}{2\pi}\rho(\omega)\frac{x^2e^x}{(e^x-1)^2}, \tag{36}$$

where $x = \hbar\omega/k_BT$ is the ratio of the vibrational energy $\hbar\omega$ and the thermal energy $k_BT$ (Fig. 13).


Published under an exclusive license by AIP Publishing

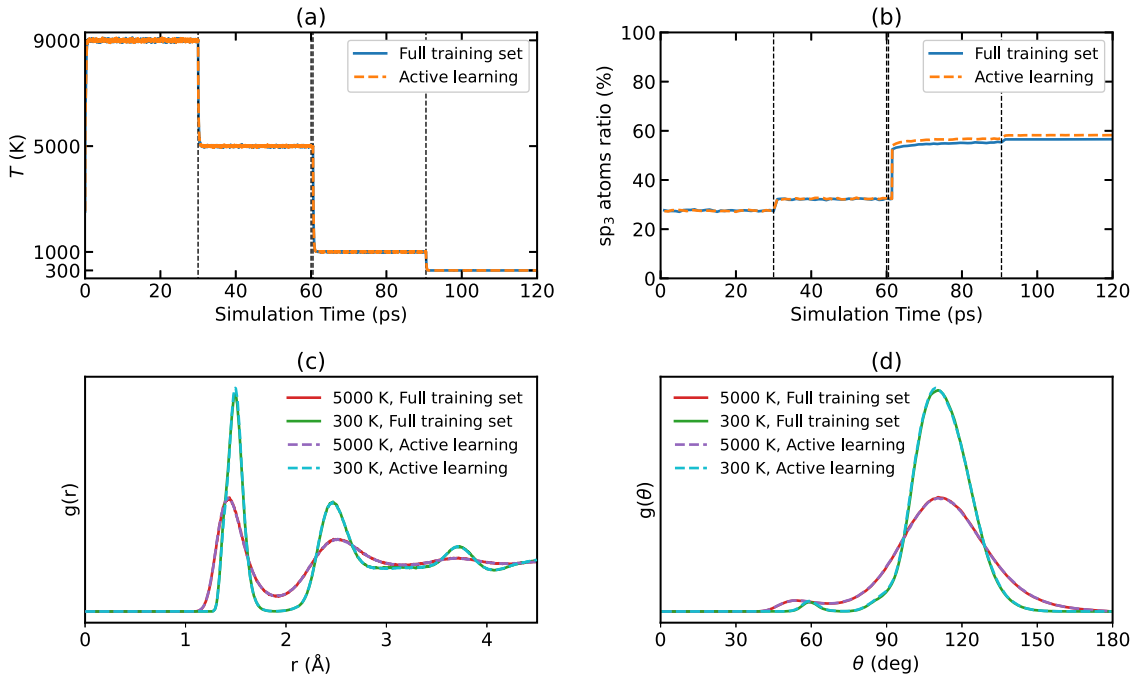


**FIG. 11.** (a) Temperature and (b) ratio of $sp^3$ bonded atoms as a function of simulation time as obtained using the NEP (4.2 Å) model. (c) Radial and (d) angular distribution functions at 5000 and 300 K.

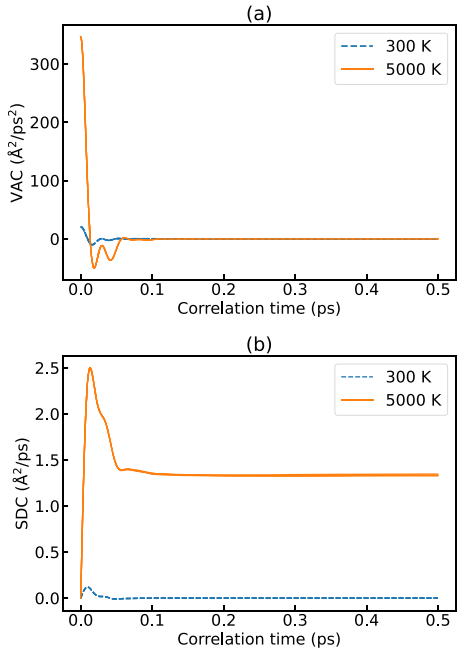


**FIG. 12.** (a) VAC and (b) running SDC as a function of correlation time for liquid carbon at 5000 K and amorphous carbon at 300 K as obtained using the NEP (4.2 Å) model. The results of five independent simulations are presented, but they are very close to each other.

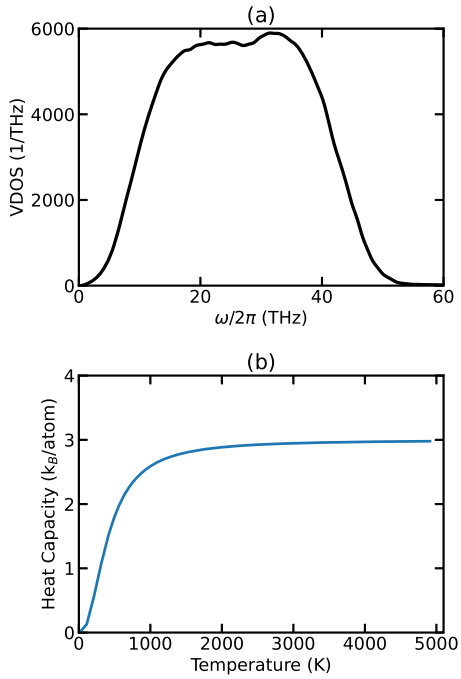


**FIG. 13.** (a) VDOS of amorphous carbon with a density of 3.0 g $cm^{-3}$ at 300 K as obtained using the NEP (4.2 Å) model. (b) Heat capacity with quantum corrections calculated from the VDOS via Eq. (36).


Published under an exclusive license by AIP Publishing

### F. Thermal conductivity of amorphous carbon

In this example, we calculate the thermal conductivity of our amorphous carbon samples using the HNEMD method and the related spectral decomposition method.[61,87] The relevant part in the `run.in` file reads as follows:

```
ensemble        nvt_nhc 300 300 100
time_step       1.0
compute_hnemd   1000 0 0 2e-4
compute_shc     5 200 2 500 400
run             2000000
```

The HNEMD simulation is invoked by the `compute_hnemd` keyword and the spectral decomposition is invoked by the `compute_shc` keyword. Thermal conductivity calculations usually require a lot of data to reduce the statistical uncertainty. To this end, we perform a number of independent runs using the above inputs. In GPUMD, the velocities are automatically initialized with different pseudo-random number seeds for different runs. Using the efficient HNEMD method, five independent runs (each with a production time of 2 ns) are sufficient to achieve high accuracy (small error bounds), as can be seen from Fig. 14(a). The thermal conductivity of the amorphous carbon structure (with a mass density of 3.0 g cm$^{-3}$) at 300 K is determined to be 5.1(1) W m$^{-1}$ K$^{-1}$, where

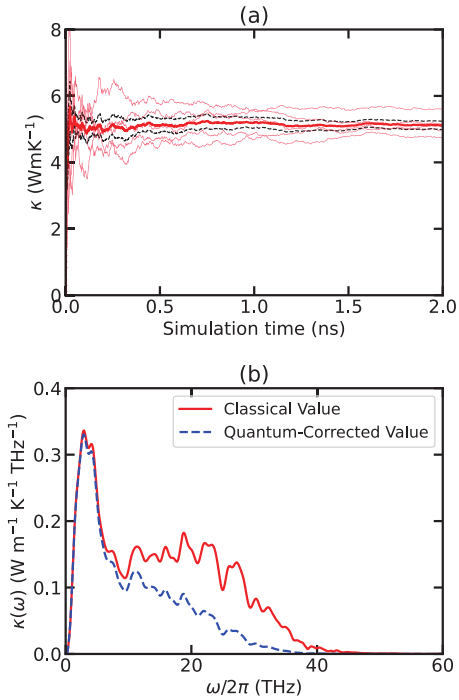


**FIG. 14.** (a) Thermal conductivity of amorphous carbon as calculated from the HNEMD method using the NEP (4.2 Å) model. The thin solid lines are from five independent runs (each with an independent initial configuration) and the thick solid line is their average. The dashed lines represent the error bounds. (b) Classical and quantum-corrected spectral thermal conductivity as a function of the vibrational frequency.

the statistical error is calculated as the standard error.[141] The thermal conductivity calculated in this way is the classical value. For disordered materials, the thermal conductivity can be quantum corrected in a way similar to the quantum correction of the heat capacity.[142] To achieve this, we first calculate the classical spectral thermal conductivity[61,87] $\kappa^{c}(\omega)$ and include quantum corrections to obtain $\kappa^{q}(\omega)$ as follows:

$$\kappa^{q}(\omega) = \kappa^{c}(\omega)\frac{x^2 e^x}{(e^x-1)^2}, \tag{37}$$

where $x = \hbar\omega/k_B T$. Both $\kappa^{c}(\omega)$ and $\kappa^{q}(\omega)$ at 300 K are shown in Fig. 14(b). The quantum-corrected thermal conductivity at 300 K is 3.2(1) W m$^{-1}$ K$^{-1}$. A more systematic investigation of the thermal transport properties in disordered carbon systems will be presented elsewhere.

### G. Tensile loading of amorphous carbon

Here, we use MD simulations to study the fracture of amorphous carbon containing 64 000 atoms with a density of 3.0 g cm$^{-3}$ under uniaxial tensile loading. The relevant part of the `run.in` input file reads as follows:

```
ensemble        npt_scr 300 300 100 0 0 0
                1000 1000 1000 1000
time_step       1.0
deform          7.52e-6 0 0 1
dump_thermo     100
dump_position   10000
run             2000000
```

The Bussi–Donadio–Parrinello thermostat[79] and the Bernetti–Bussi barostat[82] were used to control the temperature and the pressure components in the transverse ($x$ and $y$) directions. The `deform` keyword was used to deform the simulation box in the $z$ direction with a strain rate of $2 \times 10^8$ s$^{-1}$ for 2 ns, reaching up to a strain of 0.4. Based on the output thermodynamic quantities (using the `dump_thermo` keyword) and the trajectory (using the `dump_position` keyword), we can obtain the stress–strain relation and identify three representative snapshots of the fracture process as shown in Fig. 15. The fracture is ductile with an ultimate strength of about 25.0(1) GPa at a strain of about 0.1. After this point, crack initiates and grows with increasing strain, finally leading to visible void defects at a strain of 0.4.

The latent space as discussed in Sec. IV can be used to check the reliability of the tensile loading simulations. Based on the latent space of the NEP (4.2 Å) model, both the structures sampled from the trajectories of the tensile loading simulations and all those in the training dataset can be represented in a 2D PC space, as shown in Fig. 16(a). It is clear that all the structures sampled from the tensile loading simulations of the amorphous carbon are well within the region spanned by the training dataset. However, if we perform tensile loading simulations of crystalline diamond, the structures close to the fracture point are located outside of the region spanned by the training dataset. These results indicate that it is reliable to use the NEP (4.2 Å) model to study the fracture properties of amorphous carbon, while extra training samples are needed to


Published under an exclusive license by AIP Publishing

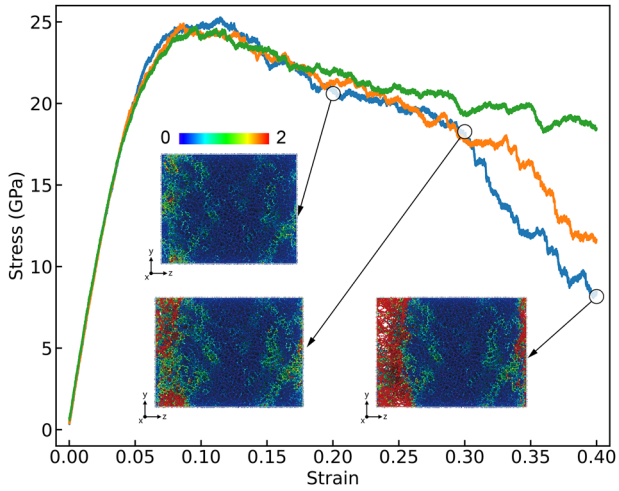


**FIG. 15.** Stress–strain relation from a uniaxial tensile loading simulation of amorphous carbon with a density of 3.0 g cm$^{-3}$ using the NEP (4.2 Å) model for carbon. The three curves correspond to three independent simulations. Three snapshots sampled at strains of 0.2, 0.3, and 0.4 during one simulation are shown as insets. The atoms in the snapshots are colored based on their atomic volumetric strain values using the OVITO package.[143]

reliably describe the fracture properties of crystalline diamond. We note by passing that using the descriptor space as shown in Fig. 16(b) fails to detect the stretched diamond structures that are identified to be distinct from all the structures in the training dataset according to the latent space. This comparison indicates that the latent space can distinguish the structures more unambiguously. Indeed, the descriptor space might contain redundant features and by passing from the descriptor space (the input layer of the neural network) to the latent space (the hidden layer), some redundant features can be eliminated, particularly owing to the application of the $\ell_1$ regularization.

## VI. INTERFACE TO OTHER CODES

A few Python packages have been developed to work with GPUMD and are briefly presented below. See the *Data availability* statement for the links of codes and documentations.

### A. The GPYUMD package

To help GPUMD users generate input and process output files, we have developed a Python interface implemented in the GPYUMD package. Reading, preparing, and writing simulation model files is facilitated by the GpumdAtoms class. This class extends the Atoms class from the popular Atomic Simulation Environment (ASE) Python package[144] to include GPUMD-specific properties. A simple example of writing a simulation model file is as follows:

```
1 from ase.lattice.cubic import Diamond
2 from gpyumd.atoms import GpumdAtoms
3
4 Si = GpumdAtoms(Diamond("Si",size
      =(10,10,10)))
5 Si.write_gpumd()
```

The GpumdAtoms class also supports adding grouping methods, sorting atoms by group or type, generating basis and $k$-point files for phonon calculations, and more.

The GPYUMD package also has a Simulation class that can be used to generate a valid simulation protocol. To do so, in addition to checking each keyword parameter, it verifies that group selections in each keyword and atom ordering in the simulation model file are consistent. As a simple demonstration, continuing from our

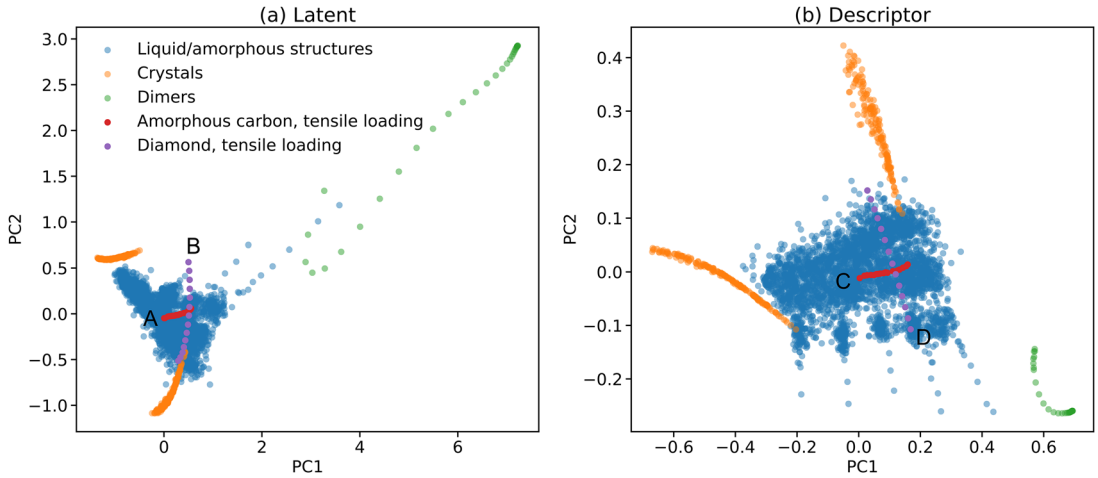


**FIG. 16.** Distribution of the training structures and some structures sampled from the tensile loading simulations in the 2D PC space (spanned by PC 1 and PC 2) as reduced from (a) the latent space and (b) the descriptor space. Note that each point here represents a structure, while each point in Fig. 8 represents an atom and the NEP models used to generate Fig. 8 and the current one are different. A–D denote the points with the maximum strain in the tensile loading simulations.


Published under an exclusive license by AIP Publishing

previous code snippet, we can create a simple simulation protocol as follows:

```
import gpyumd.keyword as kwd
from gpyumd.sim import Simulation

sim = Simulation(Si)
run = sim.add_run()
keywords = [
    kwd.Velocity(initial_temperature=300),
    kwd.TimeStep(dt_in_fs=1),
    kwd.EnsembleNVE(),
    kwd.DumpThermo(interval=1000),
    kwd.RunKeyword(number_of_steps=1e5)]

[run.add_keyword(x) for x in keywords]
ptnl = kwd.Potential(filename='Si.txt',
    symbols=['Si'])
sim.add_potential(ptnl)
sim.create_simulation()
```

The output files of GPUMD can be read and processed using simple GPYUMD functions, such as load_thermo() for thermodynamic quantity outputs and load_kappa() for thermal conductivity data from the HNEMD method. These functions return the data in convenient formats for data exploration in interactive environments such as Jupyter Notebooks.

## B. The CALORINE package

To provide a deeper integration of GPUMD within a Python workflow, we also provide the Python package CALORINE. This section provides some examples for the functionality of this package. A full documentation including extended examples and tutorials can be found at https://calorine.materialsmodeling.org/.

### 1. ASE calculator

The CALORINE package extends the functionality of ASE, implementing an ASE `Calculator` class, which lets users calculate energies, forces, and stresses with a NEP model directly from Python. Under the hood, this calculator writes and reads the GPUMD input and output files. A minimal script reads as follows:

```
from ase.build import bulk
from calorine import GPUNEP

calculator = GPUNEP('nep.txt')
atoms = bulk('Au', a=4.1)
atoms.set_calculator(calculator)
e = atoms.get_potential_energy()
```

This approach can, for example, greatly simplify the calculation of a large number of predefined structures, and it provides access to various complex structural relaxation schemes available in ASE.

### 2. Interface to GPUMD

CALORINE also interfaces directly to GPUMD using a PYBIND11 C++ interface. This enables easy access to the data structures associated with the NEP implementation in GPUMD. At the moment, the interface exposes a function for calculating the descriptors for an ASE `Atoms` object, as well as a CPU-only ASE `Calculator`. The CPU-only calculator enables using a trained NEP model on computer systems without a GPU. An example script for accessing the NEP descriptors for a structure and using the CPU-only calculator is given below.

```
from ase import Atoms
from calorine.nepy import \
    get_descriptors, CPUNEP

atoms = Atoms('CO',
      positions=[[0, 0, 0], [0, 0, 1.1]],
      cell=[20, 20, 20])
descriptors = get_descriptors(atoms)

calculator = CPUNEP('nep.txt')
atoms.set_calculator(calculator)
e = atoms.get_potential_energy()
```

## C. The PYNEP package

We also developed a Python package PYNEP to facilitate the AL process with a pre-trained NEP model. It also provides an ASE `Calculator` to calculate the properties of an `Atoms` object, including energy, forces, stress, descriptors, and latent descriptors. A simple example script for calculating these properties is as follows:

```
from ase.build import bulk
from pynep.calculate import NEP

# get energy and forces
calc = NEP('nep.txt')
atoms = bulk('C', 'diamond', cubic=True)
atoms.set_calculator(calc)
energy = atoms.get_potential_energy()
forces = atoms.get_forces()
stress = atoms.get_stress()

# get descriptors and latent descriptors
des = calc.get_property('descriptor',
    atoms)
lat = calc.get_property('latent', atoms)
```

With the descriptors or the latent descriptors available, we can select structures with different sampling methods. An example script for selecting structures using the farthest-point sampling is as follows:


Published under an exclusive license by AIP Publishing

```
from pynep.select import
    FarthestPointSample
from pynep.io import load_nep, dump_nep
import numpy as np

raw = load_nep('raw.in')
lat = np.array([np.mean(calc.get_property(
    'latent', atoms), axis=0) for atoms in
    raw])
sampler = FarthestPointSample(min_distance
    =0.05)
selected = [raw[i] for i in sampler.select
    (lat, [])]
dump_nep('selected.in', selected)
```

## VII. SUMMARY AND CONCLUSIONS

In summary, we have presented and reviewed the various features of the open-source GPUMD package, with a focus on recent developments that have enabled the generation and use of accurate and efficient NEP MLPs.[41,42] Two improvements on the atomic-environment descriptor have been introduced: one is to change the radial functions from Chebyshev basis functions to linear combinations of the basis functions and the other is to extend the angular descriptor components by considering some four-body and five-body contributions as in the ACE approach.[46] Both of these extensions are shown to improve the accuracy of NEP models further.

We have used a diverse set of materials to demonstrate that the NEP approach can achieve an above-average accuracy compared to many other state-of-the-art MLPs. In addition, NEP models can achieve a far superior computational efficiency: Typical NEP models are more than one order of magnitude faster than other MLPs and similarly more memory efficient. The high efficiency of NEP models originates from many aspects, including a reasonably small descriptor dimension (usually smaller than 100) combined with a simple neural network model with a single hidden layer, carefully derived expressions of the descriptor components, a balanced choice of the radial and angular cutoff distances, and finally a carefully optimized GPU implementation. We present our algorithms in detail in Appendix B. The latent space in the simple neural network model of NEP models also allows us to construct an effective AL scheme that can be used to greatly reduce the computational efforts in the preparation of training data.

Apart from being highly efficient, GPUMD is also user-friendly. It can both be used as a standalone package and be integrated with other packages such as ASE[144] via the GPYUMD, CALORINE, and PYNEP Python packages. The use of an efficient derivative-free optimization algorithm (SNES) greatly simplifies the implementation and excludes the dependence of GPUMD on any third-party machine learning libraries, making the installation of GPUMD effortless.

Finally, the NEP models trained using the nep executable can be directly used by the gpumd executable to perform atomistic simulations of various materials properties. To demonstrate the range of properties, length, and time scales that can be accessed via this approach, we have presented a series of examples using a NEP model trained using a standard carbon dataset.[10]

One of the disadvantages of GPUMD is that it is still not very feature-rich (as compared to similar packages such as LAMMPS[71]). However, its open-source nature and the well-designed GPU-acceleration framework have been attracting more and more developers with diverse backgrounds who are enriching the features of GPUMD at a fast pace.

## ACKNOWLEDGMENTS

Z.F. acknowledges support from the National Natural Science Foundation of China (NSFC) (Grant No. 11974059). Y.W., K.S., J.L., and H.D. acknowledge the support from the National Key Research and Development Program of China (Grant No. 2018YFB0704300). The work at Nanjing University (by J.J.W., Y.W., and J.S.) is partially supported by the NSFC (Grant Nos. 12125404, 11974162, and 11834006) and the Fundamental Research Funds for the Central Universities. The calculations performed in Nanjing University were carried out using supercomputers at the High Performance Computing Center of Collaborative Innovation Center of Advanced Microstructures, the high-performance supercomputing center of Nanjing University. P.Y. and Z.Z. acknowledge support from the NSFC (Grant No. 11932005). T.A.-N. has been supported in part by the Academy of Finland through its QTF Centre of Excellence program (Grant No. 312298) and Technology Industries of Finland Centennial Foundation Future Makers grant. Z.Z. and Y.C. gratefully acknowledge the research computing facilities offered by ITS, HKU. J.M.R., E.L., and P.E. acknowledge support from the Swedish Research Council (Grant Nos. 2018-06482, 2020-04935, and 2021-05072) and the Swedish Foundation for Strategic Research (SSF) via the SwedNess program (Grant No. GSn15-0008) as well as computational resources provided by the Swedish National Infrastructure for Computing (SNIC) at NSC, C3SE, and PDC partially funded by the Swedish Research Council (Grant No. 2018-05973).

## AUTHOR DECLARATIONS

### Conflict of Interest

The authors have no conflicts to disclose.

### Author Contributions

**Zheyong Fan**: Conceptualization (equal); Data curation (equal); Formal analysis (equal); Investigation (equal); Methodology (equal); Software (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Yanzhou Wang**: Data curation (equal); Formal analysis (equal); Investigation (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Penghua Ying**: Data curation (equal); Formal analysis (equal); Investigation (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal).- **Keke Song**: Data curation (equal); Formal analysis (equal); Investigation (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Junjie Wang**: Data curation (equal); Formal analysis (equal); Investigation (equal); Software (equal); Validation (equal); Visualization (equal);


Published under an exclusive license by AIP Publishing

Writing – original draft (equal); Writing – review & editing (equal). **Yong Wang**: Data curation (equal); Formal analysis (equal); Investigation (equal); Software (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Zezhu Zeng**: Data curation (equal); Formal analysis (equal); Investigation (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Ke Xu**: Data curation (equal); Formal analysis (equal); Investigation (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Eric Lindgren**: Data curation (equal); Formal analysis (equal); Investigation (equal); Software (equal); Validation (equal); Writing – original draft (equal); Writing – review & editing (equal). **J. Magnus Rahm**: Data curation (equal); Formal analysis (equal); Investigation (equal); Software (equal); Validation (equal); Writing – original draft (equal); Writing – review & editing (equal). **Alexander J. Gabourie**: Data curation (equal); Formal analysis (equal); Investigation (equal); Software (equal); Validation (equal); Writing – original draft (equal); Writing – review & editing (equal). **Jiahui Liu**: Data curation (equal); Formal analysis (equal); Investigation (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Haikuan Dong**: Data curation (equal); Formal analysis (equal); Investigation (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Jianyang Wu**: Resources (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal). **Yue Chen**: Resources (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal). **Zheng Zhong**: Resources (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal). **Jian Sun**: Resources (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal). **Paul Erhart**: Resources (equal); Software (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal). **Yanjing Su**: Resources (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal). **Tapio Ala-Nissila**: Resources (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal).

## DATA AVAILABILITY

The training and testing results using the various MLPs as presented in Secs. III and IV are freely available via Zenodo.[145] The input and output files for the GPUMD examples presented in Sec. V are included in the GPUMD package (https://github.com/brucefan1983/GPUMD).

The source code and documentation for GPUMD are available at https://github.com/brucefan1983/GPUMD and https://gpumd.zheyongfan.org/, respectively.

The source code and documentation for GPYUMD are available at https://github.com/AlexGabourie/gpyumd and https://gpyumd.readthedocs.io/, respectively.

The source code and documentation for CALORINE are available at https://gitlab.com/materials-modeling/calorine and https://calorine.materialsmodeling.org/, respectively.

The source code and documentation for PYNEP are available at https://github.com/bigd4/PyNEP and https://pynep.readthedocs.io/, respectively.

## APPENDIX A: SOME DERIVATIONS ON THE HEAT CURRENT EXPRESSIONS

Here, we show that the heat current expressions for the multi-body potentials as considered in Ref. 67 are special cases of the general expressions in our formulation. Without loss of generality, we take the three-body potential as an example to show this.

Using the chain rule, we first rewrite Eq. (27) as

$$\boldsymbol{J} = -\sum_i \sum_{j\neq i} \boldsymbol{r}_{ij} \frac{\partial U_i}{\partial \boldsymbol{r}_j} \cdot \boldsymbol{v}_j. \tag{A1}$$

Boone *et al.*[67] considered explicit $m$-body potentials ($m = 2, 3, 4$) that are usually used in topological force fields for organics. For the three-body potential considered therein, the site potential $U_i$ of atom $i$ is taken as the average of the potentials of the triplets it belongs to (the first index denotes the central atom of a triplet),

$$U_i = \frac{1}{3} \sum_{k\neq i} \sum_{l\neq i} (U_{ikl} + U_{kli} + U_{lik}). \tag{A2}$$

Substituting Eq. (A2) into Eq. (A1), we have

$$\boldsymbol{J} = -\frac{1}{3} \sum_i \sum_{j\neq i} \sum_{k\neq i} \sum_{l\neq i} \boldsymbol{r}_{ij} \frac{\partial (U_{ikl} + U_{kli} + U_{lik})}{\partial \boldsymbol{r}_j} \cdot \boldsymbol{v}_j. \tag{A3}$$

Note that $j$ cannot be $i$ but could be $k$ or $l$ and we, thus, have

$$\frac{\partial U_{ikl}}{\partial \boldsymbol{r}_j} = \delta_{jk} \frac{\partial U_{ijl}}{\partial \boldsymbol{r}_j} + \delta_{jl} \frac{\partial U_{ikj}}{\partial \boldsymbol{r}_j} \tag{A4}$$

and similar expressions for $\partial U_{kli}/\partial \boldsymbol{r}_j$ and $\partial U_{lik}/\partial \boldsymbol{r}_j$. Following Boone *et al.*,[67] we can define

$$\boldsymbol{F}_j^{ijl} \equiv -\frac{\partial U_{ijl}}{\partial \boldsymbol{r}_j}, \tag{A5}$$

as the force acting on atom $j$ from the triplet $ijl$. Then, we can write Eq. (A3) as

$$\boldsymbol{J} = \frac{1}{3} \sum_i \sum_{j\neq i} \sum_{k\neq i} \boldsymbol{r}_{ij} \left( \boldsymbol{F}_j^{ijk} + \boldsymbol{F}_j^{ikj} + \boldsymbol{F}_j^{jki} + \boldsymbol{F}_j^{kji} + \boldsymbol{F}_j^{jik} + \boldsymbol{F}_j^{kij} \right) \cdot \boldsymbol{v}_j. \tag{A6}$$

By manipulating the dummy indices in the summation, it can be written as

$$\begin{aligned} \boldsymbol{J} = \frac{1}{3} \sum_i \sum_{j\neq i} \sum_{k\neq i} & (\boldsymbol{r}_{ji} + \boldsymbol{r}_{ki}) \boldsymbol{F}_i^{ijk} \cdot \boldsymbol{v}_i \\ & + \left( \boldsymbol{r}_{ij} + \boldsymbol{r}_{kj} \right) \boldsymbol{F}_j^{ijk} \cdot \boldsymbol{v}_j + \left( \boldsymbol{r}_{ik} + \boldsymbol{r}_{jk} \right) \boldsymbol{F}_k^{ijk} \cdot \boldsymbol{v}_k, \end{aligned} \tag{A7}$$

which corresponds to Eq. (18d) in Ref. 67.

## APPENDIX B: ALGORITHMS

In this appendix, we present the complete algorithms for evaluating the NEP energy, force, and virial expressions as implemented in GPUMD. First, we list all the relevant quantities:

1. $N$ is the total number of atoms.
2. $\mathrm{NN}_i^{\mathrm{R}}$ is the number of neighbors of atom $i$ for the radial descriptor components.
3. $\mathrm{NN}_i^{\mathrm{A}}$ is the number of neighbors of atom $i$ for the angular descriptor components.


Published under an exclusive license by AIP Publishing

4. $\mathrm{NL}^{\mathrm{R}}_{im}$ is the index of the $m$th neighbor of atom $i$ for the radial descriptor components.
5. $\mathrm{NL}^{\mathrm{A}}_{im}$ is the index of the $m$th neighbor of atom $i$ for the angular descriptor components.
6. $\{\boldsymbol{r}_i\}_{i=0}^{N-1}$ are the atom positions.
7. $\{U_i\}_{i=0}^{N-1}$ are the site energies.
8. $\{\partial U_i/\partial q_v^i\}_{i=0}^{N-1}$ are the derivatives of the site energies with respect to the descriptor components.
9. $\{\partial U_i/\partial \boldsymbol{r}_{ij}\}_{i=0}^{N-1}$ are the partial forces.
10. $\{\boldsymbol{F}_i\}_{i=0}^{N-1}$ are the forces on the atoms.
11. $\{\mathbf{W}_i\}_{i=0}^{N-1}$ are the virials on the atoms.

## 1. Application of the neural network

We use a simple feedforward neural network with a single hidden layer. We have done extensive tests and found that a single hidden layer is sufficient to achieve high accuracy for NEP models and using more hidden layers only reduces computational performance without a corresponding improvement in model accuracy. Using a single hidden layer, many variables can be defined as registers instead of global memory or local memory in the CUDA kernel, which are much more expensive to access. Therefore, using a single hidden layer allows us to achieve a significantly higher computational performance in the parallelism scheme we adopted.

Listing 1. The function applying the feedforward neural network to the input descriptor vector to obtain the site energy of an atom and the derivative of the energy with respect to the descriptor components.

```
__device__ void apply_ann(
  const int N_des,
  const int N_neu,
  const float* w0,
  const float* b0,
  const float* w1,
  const float* b1,
  const float* q,
  float& energy,
  float* energy_derivative)
{
  for (int n = 0; n < N_neu; ++n) {
    float w0_times_q = 0.0f;
    for (int d = 0; d < N_des; ++d) {
      w0_times_q += w0[n*N_des+d] * q[d];
    }
    float x1 = tanh(w0_times_q - b0[n]);
    float tanh_der = 1.0f - x1 * x1;
    energy += w1[n] * x1;
    for (int d = 0; d < N_des; ++d) {
      float y1 = tanh_der * w0[n*N_des+d];
      energy_derivative[d] += w1[n] * y1;
    }
  }
  energy -= b1[0];
}
```

The complete __device__ function applying the neural network is presented in Listing 1. For the inputs, N_des is the dimension $N_{\mathrm{des}}$ of the descriptor vector, N_neu is the number of neurons $N_{\mathrm{neu}}$ in the hidden layer, w0 is the weight matrix $\mathbf{w}^{(0)}$, w1 is the weight vector $\mathbf{w}^{(1)}$, b0 is the bias vector $\mathbf{b}^{(0)}$ in the hidden layer, b1 is the bias $b^{(1)}$ in the output node, and q is the descriptor vector $\mathbf{q}^i$. For the outputs, `energy` is the site energy $U_i$ and `energy_derivative` is the derivative of the site energy with respect to the descriptor components $\partial U_i/\partial q_v^i$. Note that we do not need to calculate the derivative of the energy (and other related quantities such as force and virial) with respect to the neural network parameters, as required in the conventional gradient-descent approach. In our evolutionary algorithm approach, there is no need to calculate the derivative of the loss function with respect to any parameters. Therefore, our implementation is very simple regarding the neural network part; particularly, we do not make GPUMD dependent on any third-party package. This makes the installation of GPUMD very simple and straightforward.

## 2. Energy and derivative of energy with respect to descriptor

In the first CUDA kernel (see Algorithm 1), the thread associated with atom $i$ calculates the whole descriptor vector $\mathbf{q}^i$ and

**ALGORITHM 1.** Pseudo-code of the CUDA kernel for evaluating the descriptor vector $q_v^i$, the per-atom energy $U_i$, and the derivatives of the energy with respect the descriptor components $\partial U_i/\partial q_v^i$.

```
Assign atom i to CUDA thread i
if i < N then
    Read position r_i for atom i from global memory
    for m = 0 to NN_i^R − 1 do
        j ← NL_im^R
        Read in r_j from global memory and calculate r_ij
        r_ij ← minimum image of r_ij
        Calculate the radial functions g_n(r_ij) according to Eq. (3)
        Accumulate the radial descriptor components
        according to Eq. (2)
    end
    for n = 0 to n_max^A do
        for m = 0 to NN_i^A − 1 do
            j ← NL_im^A
            Read in r_j from global memory and calculate r_ij
            r_ij ← minimum image of r_ij
            Calculate the radial functions g_n(r_ij) according to
            Eq. (3) but with r_c^R changed to r_c^A
            Accumulate the summations S_n,k according to Eq. (12)
        end
        Calculate the angular descriptor components for the
        current n according to the equations in Sec. II D.
        Save the summations S_n,k for the current n to global
        memory for later use.
    end
    Apply the neural network model to get the energy U_i and
    energy derivatives ∂U_i/∂q_v^i from the descriptor q_v^i and save
    them to global memory.
end
```


Published under an exclusive license by AIP Publishing

**ALGORITHM 2.** Pseudo-code of the CUDA kernel for evaluating the force and virial from the radial descriptor components.

```
Assign atom i to CUDA thread i.
if i < N then
    Read position r_i for atom i from global memory
    for m = 0 to NN_i^R − 1 do
        j ← NL_im^R.
        Read in r_j from global memory and calculate r_ij.
        r_ij ← minimum image of r_ij.
        Calculate the partial force ∂U_i/∂r_ij related to the radial
        descriptor components, i.e., the first term on the right
        hand side of Eq. (21).
        Similarly calculate the partial force ∂U_j/∂r_ji.
        Accumulate the force F_i on atom i according to
        Eqs. (22) and (23).
        Accumulate the virial W_i on atom i according to Eq. (24).
    end
end
```

calls the `__device__` function in Listing 1 to obtain the energy $U_i$ and the derivatives $\partial U_i/\partial q_v^i$. The derivatives will be used in the next two CUDA kernels.

## 3. Force and virial from the radial descriptor components

In the second CUDA kernel (see Algorithm 2), the thread associated with atom $i$ first calculates the partial forces $\partial U_i/\partial \boldsymbol{r}_{ij}$ and $\partial U_j/\partial \boldsymbol{r}_{ji}$ related to the radial descriptor components and then accumulates the force $\boldsymbol{F}_i$ and virial $\mathbf{W}_i$ on atom $i$. For the radial descriptor components, $\partial U_i/\partial \boldsymbol{r}_{ij}$ and $\partial U_j/\partial \boldsymbol{r}_{ji}$ only differ a little; it is, thus, a good choice to calculate both within the CUDA kernel. This algorithm is very similar to that for EAM potentials[40] (which is an angular-independent many-body potential) as implemented in GPUMD.

**ALGORITHM 3.** Pseudo-code of the CUDA kernel for evaluating the partial forces $\partial U_i/\partial \boldsymbol{r}_{ij}$ for all atoms $i$ and all neighbors $j$ of $i$ from the angular descriptor components.

```
Assign atom i to CUDA thread i
if i < N then
    Read position r_i for atom i from global memory
    for m = 0 to NN_i^A − 1 do
        j ← NL_im^A
        Read in r_j from global memory and calculate r_ij
        r_ij ← minimum image of r_ij
        Calculate the partial force ∂U_i/∂r_ij related to the
        angular descriptor components, i.e., the last three
        terms on the right hand side of Eq. (21).
        Save the partial force ∂U_i/∂r_ij to global memory for
        later use.
    end
end
```

**ALGORITHM 4.** Pseudo-code of the CUDA kernel for evaluating the force and virial from the angular descriptor components.

```
Assign atom i to CUDA thread i.
if i < N then
    Read position r_i for atom i from global memory
    for m = 0 to NN_i^A − 1 do
        j ← NL_im^A.
        Read in r_j from global memory and calculate r_ij.
        r_ij ← minimum image of r_ij.
        Read in the partial force ∂U_i/∂r_ij related to the angular
        descriptor components from global memory.
        Similarly read in U_j/∂r_ji from global memory with
        some index manipulations.
        Accumulate the force F_i on atom i according to
        Eqs. (22) and (23).
        Accumulate the virial W_i on atom i according to Eq. (24).
    end
end
```

## 4. Partial forces from the angular descriptor components

In the third CUDA kernel (see Algorithm 3), the thread associated with atom $i$ calculates the partial forces $\partial U_i/\partial \boldsymbol{r}_{ij}$ related to the angular descriptor components and saves them to global memory, which is then used in the next CUDA kernel. For the angular descriptor components, $\partial U_i/\partial \boldsymbol{r}_{ij}$ and $\partial U_j/\partial \boldsymbol{r}_{ji}$ differ a lot and it is, thus, more efficient to use a two-kernel approach: (1) using one CUDA kernel (the current one) to calculate the partial forces $\{\partial U_i/\partial \boldsymbol{r}_{ij}\}$ for all atom pairs (within the angular cutoff) and save them to global memory and (2) then using another CUDA kernel (the next one) to consume them.

## 5. Force and virial from the angular partial forces

After obtaining the partial force $\{\partial U_i/\partial \boldsymbol{r}_{ij}\}$ related to the angular descriptor components, we use a CUDA kernel (see Algorithm 4) to accumulate the corresponding force and virial. In this kernel, we load the partial forces $\partial U_i/\partial \boldsymbol{r}_{ij}$ and $\partial U_j/\partial \boldsymbol{r}_{ji}$, which are related to each other by an exchange of atom indices $i$ and $j$. Then, we accumulate the force $\boldsymbol{F}_i$ on atom $i$ according to Eqs. (22) and (23) and accumulate the virial $\mathbf{W}_i$ on atom $i$ according to Eq. (24). This is a general CUDA kernel used in the GPUMD package for all the angular-dependent many-body potentials, such as the Stillinger–Weber[73] and Tersoff potentials.[38]

## REFERENCES

[1] J. Behler, "Perspective: Machine learning potentials for atomistic simulations," J. Chem. Phys. **145**, 170901 (2016).
[2] V. L. Deringer, M. A. Caro, and G. Csányi, "Machine learning interatomic potentials as emerging tools for materials science," Adv. Mater. **31**, 1902765 (2019).
[3] T. Mueller, A. Hernandez, and C. Wang, "Machine learning for interatomic potential models," J. Chem. Phys. **152**, 050902 (2020).
[4] F. Noé, A. Tkatchenko, K.-R. Müller, and C. Clementi, "Machine learning for molecular simulation," Annu. Rev. Phys. Chem. **71**, 361–390 (2020).


Published under an exclusive license by AIP Publishing

[5]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).
[6]Y. Mishin, "Machine-learning interatomic potentials for materials science," Acta Mater. **214**, 116980 (2021).
[7]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).
[8]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).
[9]A. P. Bartók, R. Kondor, and G. Csányi, "On representing chemical environments," Phys. Rev. B **87**, 184115 (2013).
[10]V. L. Deringer and G. Csányi, "Machine learning based interatomic potential for amorphous carbon," Phys. Rev. B **95**, 094203 (2017).
[11]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).
[12]A. Khorshidi and A. A. Peterson, "Amp: A modular approach to machine learning in atomistic simulations," Comput. Phys. Commun. **207**, 310–324 (2016).
[13]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).
[14]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).
[15]J. S. Smith, O. Isayev, and A. E. Roitberg, "ANI-1: An extensible neural network potential with DFT accuracy at force field computational cost," Chem. Sci. **8**, 3192–3203 (2017).
[16]K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller, "SchNet–A deep learning architecture for molecules and materials," J. Chem. Phys. **148**, 241722 (2018).
[17]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).
[18]L. Zhang, J. Han, H. Wang, R. Car, and W. E, "Deep potential molecular dynamics: A scalable model with the accuracy of quantum mechanics," Phys. Rev. Lett. **120**, 143001 (2018).
[19]L. Zhang, J. Han, H. Wang, W. A. Saidi, R. Car, and W. E, "End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems," in *Advances in Neural Information Processing Systems*, edited by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Curran Assoc. Inc., 2018), Vol. 31.
[20]K. Yao, J. E. Herr, D. Toth, R. Mckintyre, and J. Parkhill, "The TensorMol-0.1 model chemistry: A neural network augmented with long-range physics," Chem. Sci. **9**, 2261–2269 (2018).
[21]O. T. Unke and M. Meuwly, "PhysNet: A neural network for predicting energies, forces, dipole moments, and partial charges," J. Chem. Theory Comput. **15**, 3678–3693 (2019).
[22]C. Chen, W. Ye, Y. Zuo, C. Zheng, and S. P. Ong, "Graph networks as a universal machine learning framework for molecules and crystals," Chem. Mater. **31**, 3564–3572 (2019).
[23]M. A. Caro, "Optimizing many-body atomic descriptors for enhanced computational performance of machine learning based interatomic potentials," Phys. Rev. B **100**, 024112 (2019).
[24]S. Chmiela, H. E. Sauceda, I. Poltavsky, K.-R. Müller, and A. Tkatchenko, "sGDML: Constructing accurate and data efficient molecular force fields using machine learning," Comput. Phys. Commun. **240**, 38–45 (2019).
[25]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).
[26]K. Lee, D. Yoo, W. Jeong, and S. Han, "SIMPLE-NN: An efficient package for training and executing neural-network interatomic potentials," Comput. Phys. Commun. **242**, 95–103 (2019).
[27]R. Lot, F. Pellegrini, Y. Shaidu, and E. Küçükbenli, "PANNA: Properties from artificial neural network architectures," Comput. Phys. Commun. **256**, 107402 (2020).
[28]A. S. Christensen, L. A. Bratholm, F. A. Faber, and O. Anatole von Lilienfeld, "FCHL revisited: Faster and more accurate quantum machine learning," J. Chem. Phys. **152**, 044107 (2020).
[29]Y. Shao, M. Hellström, P. D. Mitev, L. Knijff, and C. Zhang, "PiNN: A Python library for building atomic neural networks of molecules and materials," J. Chem. Inf. Model. **60**, 1184–1193 (2020).
[30]A. V. Shapeev, "Moment tensor potentials: A class of systematically improvable interatomic potentials," Multiscale Model. Simul. **14**, 1153–1173 (2016).
[31]I. S. Novikov, K. Gubaev, E. V. Podryabinkin, and A. V. Shapeev, "The MLIP package: Moment tensor potentials with MPI and active learning," Mach. Learn.: Sci. Technol. **2**, 025002 (2021).
[32]Y. Zhang, J. Xia, and B. Jiang, "REANN: A PyTorch-based end-to-end multi-functional deep neural network package for molecular, reactive, and periodic systems," J. Chem. Phys. **156**, 114801 (2022).
[33]J. Byggmästar, K. Nordlund, and F. Djurabekova, "Modeling refractory high-entropy alloys with efficient machine-learned interatomic potentials: Defects and segregation," Phys. Rev. B **104**, 104101 (2021).
[34]H. Yanxon, D. Zagaceta, B. Tang, D. S. Matteson, and Q. Zhu, "PyXtal_FF: A Python library for automated force field generation," Mach. Learn.: Sci. Technol. **2**, 027001 (2021).
[35]Y. Lysogorskiy, C. van der Oord, A. Bochkarev, S. Menon, M. Rinaldi, T. Hammerschmidt, M. Mrovec, A. Thompson, G. Csányi, C. Ortner *et al.*, "Performant implementation of the atomic cluster expansion (PACE) and application to copper and silicon," npj Comput. Mater. **7**, 97 (2021).
[36]A. Bochkarev, Y. Lysogorskiy, S. Menon, M. Qamar, M. Mrovec, and R. Drautz, "Efficient parametrization of the atomic cluster expansion," Phys. Rev. Mater. **6**, 013804 (2022).
[37]M. Wen, Y. Afshar, R. S. Elliott, and E. B. Tadmor, "KLIFF: A framework to develop physics-based and machine learning interatomic potentials," Comput. Phys. Commun. **272**, 108218 (2022).
[38]J. Tersoff, "Modeling solid-state chemistry: Interatomic potentials for multicomponent systems," Phys. Rev. B **39**, 5566–5568 (1989).
[39]W. Jia, H. Wang, M. Chen, D. Lu, L. Lin, R. Car, W. E, and L. Zhang, "Pushing the limit of molecular dynamics with *ab initio* accuracy to 100 million atoms with machine learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '20* (IEEE Press, 2020).
[40]M. S. Daw and M. I. Baskes, "Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals," Phys. Rev. B **29**, 6443–6453 (1984).
[41]Z. Fan, Z. Zeng, C. Zhang, Y. Wang, K. Song, H. Dong, Y. Chen, and T. Ala-Nissila, "Neuroevolution machine learning potentials: Combining high accuracy and low cost in atomistic simulations and application to heat transport," Phys. Rev. B **104**, 104309 (2021).
[42]Z. Fan, "Improving the accuracy of the neuroevolution machine learning potential for multi-component systems," J. Phys.: Condens. Matter **34**, 125902 (2022).
[43]Z. Fan, T. Siro, and A. Harju, "Accelerated molecular dynamics force evaluation on graphics processing units for thermal conductivity calculations," Comput. Phys. Commun. **184**, 1414–1425 (2013).
[44]Z. Fan, W. Chen, V. Vierimaa, and A. Harju, "Efficient molecular dynamics simulations with many-body potentials on graphics processing units," Comput. Phys. Commun. **218**, 10–16 (2017).
[45]Y. Zhang, J. Xia, and B. Jiang, "Physically motivated recursively embedded atom neural networks: Incorporating local completeness and nonlocality," Phys. Rev. Lett. **127**, 156002 (2021).
[46]R. Drautz, "Atomic cluster expansion for accurate and transferable interatomic potentials," Phys. Rev. B **99**, 014104 (2019).
[47]D. P. Kovács, C. van der Oord, J. Kucera, A. E. A. Allen, D. J. Cole, C. Ortner, and G. Csányi, "Linear atomic cluster expansion force fields for organic molecules: Beyond RMSE," J. Chem. Theory Comput. **17**, 7696–7711 (2021).
[48]G. Dusson, M. Bachmayr, G. Csányi, R. Drautz, S. Etter, C. van der Oord, and C. Ortner, "Atomic cluster expansion: Completeness, efficiency and stability," J. Comput. Phys. **454**, 110946 (2022).
[49]J. Behler and M. Parrinello, "Generalized neural-network representation of high-dimensional potential-energy surfaces," Phys. Rev. Lett. **98**, 146401 (2007).


Published under an exclusive license by AIP Publishing

[50]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).
[51]M. F. Langer, A. Goeßmann, and M. Rupp, "Representations of molecules and materials for interpolation of quantum-mechanical simulations via machine learning," npj Comput. Mater. **8**, 41 (2022).
[52]M. Gastegger, L. Schwiedrzik, M. Bittermann, F. Berzsenyi, and P. Marquetand, "wACSF–Weighted atom-centered symmetry functions as descriptors in machine learning potentials," J. Chem. Phys. **148**, 241709 (2018).
[53]J. Nigam, S. Pozdnyakov, G. Fraux, and M. Ceriotti, "Unified theory of atom-centered representations and message-passing machine-learning schemes," J. Chem. Phys. **156**, 204115 (2022).
[54]B. Parsaeifard and S. Goedecker, "Manifolds of quasi-constant SOAP and ACSF fingerprints and the resulting failure to machine learn four-body interactions," J. Chem. Phys. **156**, 034302 (2022).
[55]S. N. Pozdnyakov, M. J. Willatt, A. P. Bartók, C. Ortner, G. Csányi, and M. Ceriotti, "Incompleteness of atomic structure representations," Phys. Rev. Lett. **125**, 166001 (2020).
[56]A. Glielmo, C. Zeni, and A. De Vita, "Efficient nonparametric *n*-body force fields from machine learning," Phys. Rev. B **97**, 184307 (2018).
[57]M. J. Willatt, F. Musil, and M. Ceriotti, "Atom-density representations for machine learning," J. Chem. Phys. **150**, 154110 (2019).
[58]M. Uhrin, "Through the eyes of a descriptor: Constructing complete, invertible descriptions of atomic environments," Phys. Rev. B **104**, 144110 (2021).
[59]A. Stone and C. Wood, "Root-rational-fraction package for exact calculation of vector-coupling coefficients," Comput. Phys. Commun. **21**, 195–205 (1980).
[60]Z. Fan, L. F. C. Pereira, H.-Q. Wang, J.-C. Zheng, D. Donadio, and A. Harju, "Force and heat current formulas for many-body potentials in molecular dynamics simulations with applications to thermal conductivity calculations," Phys. Rev. B **92**, 094301 (2015).
[61]A. J. Gabourie, Z. Fan, T. Ala-Nissila, and E. Pop, "Spectral decomposition of thermal conductivity: Comparing velocity decomposition methods in homogeneous molecular dynamics simulations," Phys. Rev. B **103**, 205421 (2021).
[62]M. Gill-Comeau and L. J. Lewis, "Heat conductivity in graphene and related materials: A time-domain modal analysis," Phys. Rev. B **92**, 195404 (2015).
[63]Z. Fan, L. F. C. Pereira, P. Hirvonen, M. M. Ervasti, K. R. Elder, D. Donadio, T. Ala-Nissila, and A. Harju, "Thermal conductivity decomposition in two-dimensional materials: Application to graphene," Phys. Rev. B **95**, 144309 (2017).
[64]K. Xu, Z. Fan, J. Zhang, N. Wei, and T. Ala-Nissila, "Thermal transport properties of single-layer black phosphorus from extensive molecular dynamics simulations," Modell. Simul. Mater. Sci. Eng. **26**, 085001 (2018).
[65]J. Brorsson, A. Hashemi, Z. Fan, E. Fransson, F. Eriksson, T. Ala-Nissila, A. V. Krasheninnikov, H.-P. Komsa, and P. Erhart, "Efficient calculation of the lattice thermal conductivity by atomistic simulations with *ab initio* accuracy," Adv. Theory Simul. **5**, 2100217 (2022).
[66]C. Verdi, F. Karsai, P. Liu, R. Jinnouchi, and G. Kresse, "Thermal transport and phase transitions of zirconia by on-the-fly machine-learned interatomic potentials," npj Comput. Mater. **7**, 156 (2021).
[67]P. Boone, H. Babaei, and C. E. Wilmer, "Heat flux for many-body interactions: Corrections to LAMMPS," J. Chem. Theory Comput. **15**, 5579–5587 (2019).
[68]T. Schaul, T. Glasmachers, and J. Schmidhuber, "High dimensions and heavy tails for natural evolution strategies," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11* (Association for Computing Machinery, New York, USA, 2011), pp. 845–852.
[69]J. A. Anderson, C. D. Lorenz, and A. Travesset, "General purpose molecular dynamics simulations fully implemented on graphics processing units," J. Comput. Phys. **227**, 5342–5359 (2008).
[70]S. Páll, A. Zhmurov, P. Bauer, M. Abraham, M. Lundborg, A. Gray, B. Hess, and E. Lindahl, "Heterogeneous parallelization and acceleration of molecular dynamics simulations in GROMACS," J. Chem. Phys. **153**, 134110 (2020).
[71]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).
[72]Z. Fan, Y. Wang, X. Gu, P. Qian, Y. Su, and T. Ala-Nissila, "A minimal Tersoff potential for diamond silicon with improved descriptions of elastic and phonon transport properties," J. Phys.: Condens. Matter **32**, 135901 (2019).
[73]F. H. Stillinger and T. A. Weber, "Computer simulation of local order in condensed phases of silicon," Phys. Rev. B **31**, 5262–5271 (1985).
[74]F. Eriksson, E. Fransson, and P. Erhart, "The hiPhive package for the extraction of high-order force constants by machine learning," Adv. Theory Simul. **2**, 1800184 (2019).
[75]H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak, "Molecular dynamics with coupling to an external bath," J. Chem. Phys. **81**, 3684–3690 (1984).
[76]S. Nosé, "A unified formulation of the constant temperature molecular dynamics methods," J. Chem. Phys. **81**, 511–519 (1984).
[77]W. G. Hoover, "Canonical dynamics: Equilibrium phase-space distributions," Phys. Rev. A **31**, 1695–1697 (1985).
[78]G. J. Martyna, M. L. Klein, and M. Tuckerman, "Nosé-Hoover chains: The canonical ensemble via continuous dynamics," J. Chem. Phys. **97**, 2635–2643 (1992).
[79]G. Bussi, D. Donadio, and M. Parrinello, "Canonical sampling through velocity rescaling," J. Chem. Phys. **126**, 014101 (2007).
[80]G. Bussi and M. Parrinello, "Accurate sampling using Langevin dynamics," Phys. Rev. E **75**, 056707 (2007).
[81]B. Leimkuhler and C. Matthews, "Robust and efficient configurational molecular sampling via Langevin dynamics," J. Chem. Phys. **138**, 174102 (2013).
[82]M. Bernetti and G. Bussi, "Pressure control using stochastic cell rescaling," J. Chem. Phys. **153**, 114107 (2020).
[83]M. S. Green, "Markoff random processes and the statistical mechanics of time-dependent phenomena. II. Irreversible processes in fluids," J. Chem. Phys. **22**, 398–413 (1954).
[84]R. Kubo, "Statistical-mechanical theory of irreversible processes. I. General theory and simple applications to magnetic and conduction problems," J. Phys. Soc. Jpn. **12**, 570–586 (1957).
[85]Z. Li, S. Xiong, C. Sievers, Y. Hu, Z. Fan, N. Wei, H. Bao, S. Chen, D. Donadio, and T. Ala-Nissila, "Influence of thermostatting on nonequilibrium molecular dynamics simulations of heat conduction in solids," J. Chem. Phys. **151**, 234105 (2019).
[86]D. J. Evans, "Homogeneous NEMD algorithm for thermal conductivity—Application of non-canonical linear response theory," Phys. Lett. A **91**, 457–460 (1982).
[87]Z. Fan, H. Dong, A. Harju, and T. Ala-Nissila, "Homogeneous nonequilibrium molecular dynamics method for heat transport and spectral decomposition with many-body potentials," Phys. Rev. B **99**, 064308 (2019).
[88]H. Dong, Z. Fan, L. Shi, A. Harju, and T. Ala-Nissila, "Equivalence of the equilibrium and the nonequilibrium molecular dynamics methods for thermal conductivity calculations: From bulk to nanowire silicon," Phys. Rev. B **97**, 094305 (2018).
[89]B. Mortazavi, Z. Fan, L. F. C. Pereira, A. Harju, and T. Rabczuk, "Amorphized graphene: A stiff material with low thermal conductivity," Carbon **103**, 318–326 (2016).
[90]K. Azizi, P. Hirvonen, Z. Fan, A. Harju, K. R. Elder, T. Ala-Nissila, and S. M. V. Allaei, "Kapitza thermal resistance across individual grain boundaries in graphene," Carbon **125**, 384–390 (2017).
[91]Z. Fan, P. Hirvonen, L. F. C. Pereira, M. M. Ervasti, K. R. Elder, D. Donadio, A. Harju, and T. Ala-Nissila, "Bimodal grain-size scaling of thermal transport in polycrystalline graphene from large-scale molecular dynamics simulations," Nano Lett. **17**, 5919–5924 (2017).
[92]B. Mortazavi, A. Lherbier, Z. Fan, A. Harju, T. Rabczuk, and J.-C. Charlier, "Thermal and electronic transport characteristics of highly stretchable graphene kirigami," Nanoscale **9**, 16329–16341 (2017).
[93]B. Mortazavi, M. Makaremi, M. Shahrokhi, Z. Fan, and T. Rabczuk, "N-graphdiyne two-dimensional nanomaterials: Semiconductors with low thermal conductivity and high stretchability," Carbon **137**, 57–67 (2018).


Published under an exclusive license by AIP Publishing

[94]H. Dong, P. Hirvonen, Z. Fan, and T. Ala-Nissila, "Heat transport in pristine and polycrystalline single-layer hexagonal boron nitride," Phys. Chem. Chem. Phys. **20**, 24602–24612 (2018).
[95]K. Xu, A. J. Gabourie, A. Hashemi, Z. Fan, N. Wei, A. B. Farimani, H.-P. Komsa, A. V. Krasheninnikov, E. Pop, and T. Ala-Nissila, "Thermal transport in $MoS_2$ from molecular dynamics using different empirical potentials," Phys. Rev. B **99**, 054303 (2019).
[96]X. Gu, Z. Fan, H. Bao, and C. Y. Zhao, "Revisiting phonon-phonon scattering in single-layer graphene," Phys. Rev. B **100**, 064306 (2019).
[97]X. Wu and Q. Han, "Thermal conductivity of monolayer hexagonal boron nitride: From defective to amorphous," Comput. Mater. Sci. **184**, 109938 (2020).
[98]X. Wu and Q. Han, "Thermal conductivity of defective graphene: An efficient molecular dynamics study based on graphics processing units," Nanotechnology **31**, 215708 (2020).
[99]X. Wu and Q. Han, "Semidefective graphene/h-BN in-plane heterostructures: Enhancing interface thermal conductance by topological defects," J. Phys. Chem. C **125**, 2748–2760 (2021).
[100]X. Wu and Q. Han, "Thermal transport in pristine and defective two-dimensional polyaniline ($C_3N$)," Int. J. Heat Mass Transfer **173**, 121235 (2021).
[101]H. Dong, P. Hirvonen, Z. Fan, P. Qian, Y. Su, and T. Ala-Nissila, "Heat transport across graphene/hexagonal-BN tilted grain boundaries from phase-field crystal model and molecular dynamics simulations," J. Appl. Phys. **130**, 235102 (2021).
[102]P. Ying, T. Liang, Y. Du, J. Zhang, X. Zeng, and Z. Zhong, "Thermal transport in planar $sp^2$-hybridized carbon allotropes: A comparative study of biphenylene network, pentaheptite and graphene," Int. J. Heat Mass Transfer **183**, 122060 (2022).
[103]W. Sha and F. Guo, "Thermal transport in two-dimensional carbon nitrides: A comparative molecular dynamics study," Carbon Trends **7**, 100161 (2022).
[104]A. Rajabpour, Z. Fan, and S. M. Vaez Allaei, "Inter-layer and intra-layer heat transfer in bilayer/monolayer graphene van der Waals heterostructure: Is there a Kapitza resistance analogous?," Appl. Phys. Lett. **112**, 233104 (2018).
[105]A. J. Gabourie, S. V. Suryavanshi, A. B. Farimani, and E. Pop, "Reduced thermal conductivity of supported and encased monolayer and bilayer $MoS_2$," 2D Mater. **8**, 011001 (2020).
[106]S. E. Kim, F. Mujid, A. Rai, F. Eriksson, J. Suh, P. Poddar, A. Ray, C. Park, E. Fransson, Y. Zhong, D. A. Muller, P. Erhart, D. G. Cahill, and J. Park, "Extremely anisotropic van der Waals thermal conductors," Nature **597**, 660–665 (2021).
[107]X. Wu and Q. Han, "Phonon thermal transport across multilayer graphene/hexagonal boron nitride van der Waals heterostructures," ACS Appl. Mater. Interfaces **13**, 32564–32578 (2021).
[108]X. Wu and Q. Han, "Transition from incoherent to coherent phonon thermal transport across graphene/h-BN van der Waals superlattices," Int. J. Heat Mass Transfer **184**, 122390 (2022).
[109]X. Wu and Q. Han, "Maximum thermal conductivity of multilayer graphene with periodic two-dimensional empty space," Int. J. Heat Mass Transfer **191**, 122829 (2022).
[110]H. Feng, K. Zhang, X. Wang, G. Zhang, and X. Guo, "Thermal transport of bilayer graphene: A homogeneous nonequilibrium molecular dynamics study," Phys. Scr. **97**, 045704 (2022).
[111]H. Dong, Z. Fan, P. Qian, T. Ala-Nissila, and Y. Su, "Thermal conductivity reduction in carbon nanotube by fullerene encapsulation: A molecular dynamics study," Carbon **161**, 800–808 (2020).
[112]G. Barbalinardo, Z. Chen, H. Dong, Z. Fan, and D. Donadio, "Ultrahigh convergent thermal conductivity of carbon nanotubes from comprehensive atomistic modeling," Phys. Rev. Lett. **127**, 025902 (2021).
[113]T. Liang, K. Xu, M. Han, Y. Yao, Z. Zhang, X. Zeng, J. Xu, and J. Wu, "Abnormally high thermal conductivity in fivefold twinned diamond nanowires," Mater. Today Phys. **25**, 100705 (2022).
[114]L. Isaeva, G. Barbalinardo, D. Donadio, and S. Baroni, "Modeling heat transport in crystals and glasses from a unified lattice-dynamical approach," Nat. Commun. **10**, 3853 (2019).
[115]B. Fu, K. D. Parrish, H.-Y. Kim, G. Tang, and A. J. H. McGaughey, "Phonon confinement and transport in ultrathin films," Phys. Rev. B **101**, 045417 (2020).
[116]Z. Zhang, Y. Guo, M. Bescond, J. Chen, M. Nomura, and S. Volz, "Generalized decay law for particlelike and wavelike thermal phonons," Phys. Rev. B **103**, 184307 (2021).
[117]H. Wang, Y. Cheng, Z. Fan, Y. Guo, Z. Zhang, M. Bescond, M. Nomura, T. Ala-Nissila, S. Volz, and S. Xiong, "Anomalous thermal conductivity enhancement in low dimensional resonant nanostructures due to imperfections," Nanoscale **13**, 10010–10015 (2021).
[118]H. Dong, S. Xiong, Z. Fan, P. Qian, Y. Su, and T. Ala-Nissila, "Interpretation of apparent thermal conductivity in finite systems from equilibrium molecular dynamics simulations," Phys. Rev. B **103**, 035417 (2021).
[119]N. W. Lundgren, G. Barbalinardo, and D. Donadio, "Mode localization and suppressed heat transport in amorphous alloys," Phys. Rev. B **103**, 024204 (2021).
[120]K. Li, Y. Cheng, H. Wang, Y. Guo, Z. Zhang, M. Bescond, M. Nomura, S. Volz, X. Zhang, and S. Xiong, "Phonon resonant effect in silicon membranes with different crystallographic orientations," Int. J. Heat Mass Transfer **183**, 122144 (2022).
[121]H. Dong, Z. Fan, P. Qian, and Y. Su, "Exactly equivalent thermal conductivity in finite systems from equilibrium and nonequilibrium molecular dynamics simulations," Physica E **144**, 115410 (2022).
[122]S. Jin, Z. Zhang, Y. Guo, J. Chen, M. Nomura, and S. Volz, "Optimization of interfacial thermal transport in Si/Ge heterostructure driven by machine learning," Int. J. Heat Mass Transfer **182**, 122014 (2022).
[123]W. Jiang, Y. Zhang, L. Zhang, and H. Wang, "Accurate Deep Potential model for the Al–Cu–Mg alloy in the full concentration space," Chin. Phys. B **30**, 050706 (2021).
[124]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).
[125]J. Nigam, S. Pozdnyakov, and M. Ceriotti, "Recursive evaluation and iterative contraction of *N*-body equivariant features," J. Chem. Phys. **153**, 121101 (2020).
[126]S. Chmiela, A. Tkatchenko, H. E. Sauceda, I. Poltavsky, K. T. Schütt, and K.-R. Müller, "Machine learning of accurate energy-conserving molecular force fields," Sci. Adv. **3**, e1603015 (2017).
[127]A. S. Christensen and O. A. von Lilienfeld, "On the role of gradients for machine learning of molecular energies and forces," Mach. Learn.: Sci. Technol. **1**, 045018 (2020).
[128]M. Gastegger, J. Behler, and P. Marquetand, "Machine learning molecular dynamics for the simulation of infrared spectra," Chem. Sci. **8**, 6924–6935 (2017).
[129]J. S. Smith, B. Nebgen, N. Lubbers, O. Isayev, and A. E. Roitberg, "Less is more: Sampling chemical space with active learning," J. Chem. Phys. **148**, 241733 (2018).
[130]L. Zhang, D.-Y. Lin, H. Wang, R. Car, and W. E, "Active learning of uniformly accurate interatomic potentials for materials simulation," Phys. Rev. Mater. **3**, 023804 (2019).
[131]M. Wen and E. B. Tadmor, "Uncertainty quantification in molecular simulations with dropout neural network potentials," npj Comput. Mater. **6**, 124 (2020).
[132]C. Schran, J. Behler, and D. Marx, "Automated fitting of neural network potentials at coupled cluster accuracy: Protonated water clusters as testing ground," J. Chem. Theory Comput. **16**, 88–99 (2020).
[133]M. Karabin and D. Perez, "An entropy-maximization approach to automated training set generation for interatomic potentials," J. Chem. Phys. **153**, 094110 (2020).
[134]E. V. Podryabinkin and A. V. Shapeev, "Active learning of linearly parametrized interatomic potentials," Comput. Mater. Sci. **140**, 171–180 (2017).
[135]V. Zaverkin and J. Kästner, "Exploration of transferable and uniformly accurate neural network interatomic potentials using optimal experimental design," Mach. Learn.: Sci. Technol. **2**, 035009 (2021).


Published under an exclusive license by AIP Publishing

[136]J. P. Janet, C. Duan, T. Yang, A. Nandy, and H. J. Kulik, "A quantitative uncertainty metric controls error in neural network-driven chemical discovery," Chem. Sci. **10**, 7913–7922 (2019).
[137]K. Shimamura, Y. Takeshita, S. Fukushima, A. Koura, and F. Shimojo, "Computational and training requirements for interatomic potential based on artificial neural network for estimating low thermal conductivity of silver chalcogenides," J. Chem. Phys. **153**, 234301 (2020).
[138]M. A. Caro, V. L. Deringer, J. Koskinen, T. Laurila, and G. Csányi, "Growth mechanism and origin of high $sp^3$ content in tetrahedral amorphous carbon," Phys. Rev. Lett. **120**, 166101 (2018).
[139]Y. Wang, Z. Fan, P. Qian, T. Ala-Nissila, and M. A. Caro, "Structure and pore size distribution in nanoporous carbon," Chem. Mater. **34**, 617–628 (2022).
[140]J. M. Dickey and A. Paskin, "Computer simulation of the lattice dynamics of solids," Phys. Rev. **188**, 1407–1418 (1969).
[141]J. M. Haile, *Molecular Dynamics Simulation: Elementary Methods* (John Wiley & Sons, 1992).
[142]X. Gu, Z. Fan, and H. Bao, "Thermal conductivity prediction by atomistic simulation methods: Recent advances and detailed comparison," J. Appl. Phys. **130**, 210902 (2021).
[143]A. Stukowski, "Visualization and analysis of atomistic simulation data with OVITO—The Open Visualization Tool," Modell. Simul. Mater. Sci. Eng. **18**, 015012 (2009).
[144]A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. B. Jensen, J. Kermode, J. R. Kitchin, E. L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen, "The atomic simulation environment—A Python library for working with atoms," J. Phys.: Condens. Matter **29**, 273002 (2017).
[145]Z. Fan (2022). "GPUMD: A package for constructing accurate machine-learned potentials and performing highly efficient atomistic simulations," Zenodo.10.5281/zenodo.6548090.


Published under an exclusive license by AIP Publishing