The Journal of Chemical Physics

RESEARCH ARTICLE | AUGUST 01 2023

DeePMD-kit v2: A software package for deep potential models

Jinzhe Zeng ; Duo Zhang ; Denghui Lu ; Pinghui Mo ; Zeyu Li ; Yixiao Chen ; Marián Rynik ; Li'ang Huang ; Ziyao Li ; Shaochen Shi ; Yingze Wang; Haotian Ye ; Ping Tuo ; Jiabin Yang; Ye Ding ; Yifan Li ; Davide Tisi ; Qiyu Zeng ; Han Bao ; Yu Xia ; Jiameng Huang ; Koki Muraoka ; Yibo Wang ; Junhan Chang ; Fengbo Yuan ; Sigbjørn Løland Bore ; Chun Cai ; Yinnian Lin ; Bo Wang ; Jiayan Xu ; Jia-Xin Zhu ; Chenxing Luo ; Yuzhi Zhang ; Rhys E. A. Goodall ; Wenshuo Liang ; Anurag Kumar Singh ; Sikai Yao ; Jingchao Zhang ; Renata Wentzcovitch ; Jiequn Han ; Jie Liu ; Weile Jia ; Darrin M. York ; Weinan E ; Roberto Car ; Linfeng Zhang ; Han Wang

J. Chem. Phys. 159, 054801 (2023)
https://doi.org/10.1063/5.0155600

The Journal of Chemical Physics ARTICLE pubs.aip.org/aip/jcp

DeePMD-kit v2: A software package for deep potential models

Cite as: J. Chem. Phys. 159, 054801 (2023); doi: 10.1063/5.0155600
Submitted: 21 April 2023 • Accepted: 3 July 2023 •
Published Online: 1 August 2023

Jinzhe Zeng,[1] Duo Zhang,[2,3,4] Denghui Lu,[5] Pinghui Mo,[6] Zeyu Li,[7] Yixiao Chen,[8] Marián Rynik,[9] Li'ang Huang,[10] Ziyao Li,[3,11] Shaochen Shi,[12] Yingze Wang,[3,13] Haotian Ye,[7] Ping Tuo,[2] Jiabin Yang,[14] Ye Ding,[15,16] Yifan Li,[17] Davide Tisi,[18,19] Qiyu Zeng,[20] Han Bao,[21,22] Yu Xia,[12] Jiameng Huang,[3,23] Koki Muraoka,[24] Yibo Wang,[3] Junhan Chang,[3,13] Fengbo Yuan,[3] Sigbjørn Løland Bore,[25] Chun Cai,[2,3] Yinnian Lin,[26] Bo Wang,[27] Jiayan Xu,[28] Jia-Xin Zhu,[29] Chenxing Luo,[30] Yuzhi Zhang,[3] Rhys E. A. Goodall,[31] Wenshuo Liang,[3] Anurag Kumar Singh,[32] Sikai Yao,[3] Jingchao Zhang,[33] Renata Wentzcovitch,[30,34] Jiequn Han,[35] Jie Liu,[6] Weile Jia,[21,22] Darrin M. York,[1] Weinan E,[2,36] Roberto Car,[17] Linfeng Zhang,[2,3,a)] and Han Wang[5,37,b)]

AFFILIATIONS

[1]Laboratory for Biomolecular Simulation Research, Institute for Quantitative Biomedicine and Department of Chemistry and Chemical Biology, Rutgers University, Piscataway, New Jersey 08854, USA
[2]AI for Science Institute, Beijing 100080, People's Republic of China
[3]DP Technology, Beijing 100080, People's Republic of China
[4]Academy for Advanced Interdisciplinary Studies, Peking University, Beijing 100871, People's Republic of China
[5]HEDPS, CAPT, College of Engineering, Peking University, Beijing 100871, People's Republic of China
[6]College of Electrical and Information Engineering, Hunan University, Changsha, People's Republic of China
[7]Yuanpei College, Peking University, Beijing 100871, People's Republic of China
[8]Program in Applied and Computational Mathematics, Princeton University, Princeton, New Jersey 08540, USA
[9]Department of Experimental Physics, Comenius University, Mlynská Dolina F2, 842 48 Bratislava, Slovakia
[10]Center for Quantum Information, Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing 100084, People's Republic of China
[11]Center for Data Science, Peking University, Beijing 100871, People's Republic of China
[12]ByteDance Research, Zhonghang Plaza, No. 43, North 3rd Ring West Road, Haidian District, Beijing, People's Republic of China
[13]College of Chemistry and Molecular Engineering, Peking University, Beijing 100871, People's Republic of China
[14]Baidu, Inc., Beijing, People's Republic of China
[15]Key Laboratory of Structural Biology of Zhejiang Province, School of Life Sciences, Westlake University, Hangzhou, Zhejiang, People's Republic of China
[16]Westlake AI Therapeutics Lab, Westlake Laboratory of Life Sciences and Biomedicine, Hangzhou, Zhejiang, People's Republic of China
[17]Department of Chemistry, Princeton University, Princeton, New Jersey 08544, USA
[18]SISSA, Scuola Internazionale Superiore di Studi Avanzati, 34136 Trieste, Italy
[19]Laboratory of Computational Science and Modeling, Institute of Materials, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland
[20]Department of Physics, National University of Defense Technology, Changsha, Hunan 410073, People's Republic of China
[21]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, People's Republic of China
[22]University of Chinese Academy of Sciences, Beijing, People's Republic of China
[23]School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, People's Republic of China

[24]Department of Chemical System Engineering, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
[25]Hylleraas Centre for Quantum Molecular Sciences and Department of Chemistry, University of Oslo, P.O. Box 1033 Blindern, 0315 Oslo, Norway
[26]Wangxuan Institute of Computer Technology, Peking University, Beijing 100871, People's Republic of China
[27]Shanghai Engineering Research Center of Molecular Therapeutics and New Drug Development, Shanghai Key Laboratory of Green Chemistry and Chemical Process, School of Chemistry and Molecular Engineering, East China Normal University, Shanghai 200062, People's Republic of China
[28]School of Chemistry and Chemical Engineering, Queen's University Belfast, Belfast BT9 5AG, United Kingdom
[29]State Key Laboratory of Physical Chemistry of Solid Surfaces, iChEM, College of Chemistry and Chemical Engineering, Xiamen University, Xiamen 361005, People's Republic of China
[30]Department of Applied Physics and Applied Mathematics, Columbia University, New York, New York 10027, USA
[31]Independent Researcher, London, United Kingdom
[32]Department of Data Science, Indian Institute of Technology, Palakkad, Kerala, India
[33]NVIDIA AI Technology Center (NVAITC), Santa Clara, California 95051, USA
[34]Department of Earth and Environmental Sciences, Columbia University, New York, New York 10027, USA
[35]Center for Computational Mathematics, Flatiron Institute, New York, New York 10010, USA
[36]Center for Machine Learning Research and School of Mathematical Sciences, Peking University, Beijing 100871, People's Republic of China
[37]Laboratory of Computational Physics, Institute of Applied Physics and Computational Mathematics, Fenghao East Road 2, Beijing 100094, People's Republic of China

Note: This paper is part of the JCP Special Topic on Software for Atomistic Machine Learning.
[a)]**Electronic mail:** linfeng.zhang.zlf@gmail.com
[b)]**Author to whom correspondence should be addressed:** wang_han@iapcm.ac.cn

ABSTRACT

DeePMD-kit is a powerful open-source software package that facilitates molecular dynamics simulations using machine learning potentials known as Deep Potential (DP) models. This package, which was released in 2017, has been widely used in the fields of physics, chemistry, biology, and material science for studying atomistic systems. The current version of DeePMD-kit offers numerous advanced features, such as DeepPot-SE, attention-based and hybrid descriptors, the ability to fit tensile properties, type embedding, model deviation, DP-range correction, DP long range, graphics processing unit support for customized operators, model compression, non-von Neumann molecular dynamics, and improved usability, including documentation, compiled binary packages, graphical user interfaces, and application programming interfaces. This article presents an overview of the current major version of the DeePMD-kit package, highlighting its features and technical details. Additionally, this article presents a comprehensive procedure for conducting molecular dynamics as a representative application, benchmarks the accuracy and efficiency of different models, and discusses ongoing developments.

 https://doi.org/10.1063/5.0155600

I. INTRODUCTION

In recent years, the increasing popularity of machine learning potentials (MLPs) has revolutionized molecular dynamics (MD) simulations across various fields, including neural network potentials (NNPs),[1–19] message passing models,[7,20–24] and other machine learning models.[25–28] Numerous software packages have been developed to support the use of MLPs.[13,29–40] One of the main reasons for the widespread adoption of MLPs is their exceptional speed and accuracy, which outperform traditional molecular mechanics (MM) and *ab initio* quantum mechanics (QM) methods.[41,42] As a result, MLP-powered MD simulations have become ubiquitous in the field and are increasingly recognized as a valuable tool for studying atomistic systems.[43–49]

The DeePMD-kit is an open-source software package that facilitates molecular dynamics (MD) simulations using neural network potentials. The package was first released in 2017[29] and has since undergone rapid development with contributions from many developers. The DeePMD-kit implements a series of MLP models known as Deep Potential (DP) models,[9,10,50–54] which have been widely adopted in the fields of physics, chemistry, biology, and material science for studying a broad range of atomistic systems. These systems include metallic materials,[55] non-metallic inorganic materials,[56–60] water,[61–71] organic systems,[10,72] solutions,[52,73–76] gas-phase systems,[77–80] macromolecular systems,[81,82] and interfaces.[83–87] Furthermore, the DeePMD-kit is capable of simulating systems containing almost all Periodic Table elements,[51] operating under a wide range of temperature and pressure,[88] and can handle drug-like

molecules,[72,89] ions,[73,76] transition states,[75,77] and excited states.[90] As a result, the DeePMD-kit is a powerful and versatile tool that can be used to simulate a wide range of atomistic systems. Here, we present three exemplary instances that highlight its diverse applications.

Theoretical investigation of the water phase diagram poses a significant challenge due to the requirement for a highly accurate model of water interatomic interactions.[91,92] Consequently, it serves as an exceptionally stringent test for the model's accuracy and provides a means to validate the software implementation necessary for molecular dynamics simulations used in phase diagram calculations.[92] Zhang *et al.*[88] utilized DeePMD-kit to construct a deep potential model for the water system, covering a range of thermodynamic states from 0 to 2400 K and 0–50 GPa. The model was trained on density functional theory (DFT) data generated using the SCAN approximation of the exchange–correlation functional and exhibited consistent accuracy [with an Root mean square error (RMSE) of less than 1 meV/H2O] within the relevant thermodynamic range. Moreover, it accurately predicted fluid, molecular, and ionic phases and all stable ice polymorphs within the range, except for phases III and XV. The study extensively investigated the two first-order phase transitions from ice VII to VII" and VII" to ionic fluid and the atomistic mechanism of proton diffusion, leveraging the model's capability and high accuracy in predicting water molecule ionization.

Another challenging area is condensed-phase MD simulations, as long-range interactions are critical for modeling heterogeneous systems in the condensed phase. Electrostatic interactions are not only the longest but also are well-understood, and linear-scaling methods exist for their efficient computation at the point charge,[93] multipole,[94,95] and quantum mechanical[96,97] levels. Fast semiempirical quantum mechanical methods can be developed[98,99] that can accurately and efficiently model charge densities and many-body effects in the long-range but may still lack quantitative accuracy in the mid-range (typically less than 8 Å). This limits the predictive capability of the methods in condensed-phase simulations. Zeng *et al.*[52] created a new Δ-MLP method called Deep Potential-Range correction (DPRc) to integrate with combined quantum mechanical/molecular mechanical (QM/MM) potentials, which corrects the potential energy from a fast, linear-scaling low-level semiempirical QM/MM theory to a high-level *ab initio* QM/MM theory. Unlike many of the emerging Δ-MLPs that correct internal QM energy and forces, the DPRc model corrects both the QM–QM and QM–MM interactions of a QM/MM calculation in a manner that conserves energy as MM atoms enter (or leave) the vicinity of the QM region. This enables the model to be easily integrated as a mid-ranged correction to the potential energy within molecular simulation software that uses non-bonded lists, i.e., for each atom, a list of other atoms within a fixed cut-off distance (typically 8–12 Å). The trained DPRc model with a 6 Å range-correction was applied to simulate RNA 2′-O-transphosphorylation reactions in solution in long timescales[75] and obtain better free energy estimates with the help of the generalization of the weighted thermodynamic perturbation (gwTP) method.[100] Very recently, Zeng *et al.*[72] have trained a Δ-MLP correction model called Quantum Deep Potential Interaction (QDπ) for drug-like molecules, including tautomeric forms and protonation states, which was found to be superior to other semiempirical methods and pure MLP models.[89]

The third important application is large-scale reactive MD simulations over a nanosecond time scale, which enable the construction of interwoven reaction networks for complex reactive systems[101] instead of focusing on studying a single reaction. These simulations require the potential energy model to be accurate and computationally efficient, covering the chemical space of possible reactions. Zeng *et al.*[77] introduced a deep potential model for simulating 1 ns methane combustion reactions and identified 798 different chemical reactions in both space and time using the ReacNetGenerator package.[102] The concurrent learning procedure[103] was adopted and proved crucial in exploring known and unknown chemical space during the complex reaction process. Subsequent work conducted by the research team extended these simulations to more complex reactive systems, including linear alkane pyrolysis,[78] decomposition of explosive,[79,104] and the growth of polycyclic aromatic hydrocarbon.[80]

Compared to its initial release,[29] DeePMD-kit has evolved significantly, with the current version (v2.2.1) offering an extensive range of features. These include DeepPot-SE, attention-based, and hybrid descriptors,[10,50,51,53] the ability to fit tensorial properties,[105,106] type embedding, model deviation,[103,107] Deep Potential-Range Correction (DPRc),[52,75] Deep Potential Long Range (DPLR),[53] graphics processing unit (GPU) support for customized operators,[108] model compression,[109] non-von Neumann molecular dynamics (NVNMD),[110] and various usability improvements, such as documentation, compiled binary packages, graphical user interfaces (GUIs), and application programming interfaces (APIs). This article provides an overview of the current major version of the DeePMD-kit, highlighting its features and technical details, presenting a comprehensive procedure for conducting molecular dynamics as a representative application, benchmarking the accuracy and efficiency of different models, and discussing ongoing developments.

II. FEATURES

In this section, we introduce features from the perspective of components (shown in Fig. 1). A component represents units of computation. It is organized as a Python class inside the package, and a corresponding TensorFlow static graph will be created at runtime.

A. Models

A Deep Potential (DP) model, denoted by $\mathcal{M}$, can be generally represented as

$$\boldsymbol{y}_i = \mathcal{M}(\boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j\in n(i)}; \boldsymbol{\theta}) = \mathcal{F}\big(\mathcal{D}(\boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j\in n(i)}; \boldsymbol{\theta}_d); \boldsymbol{\theta}_f\big), \quad (1)$$

where $\boldsymbol{y}_i$ is the fitting properties, $\mathcal{F}$ is the fitting network (introduced in Sec. II A 3), and $\mathcal{D}$ is the descriptor (introduced in Sec. II A 2). $\boldsymbol{x} = (\boldsymbol{r}_i, \alpha_i)$, with $\boldsymbol{r}_i$ being the Cartesian coordinates and α_i being the chemical species, denotes the degrees of freedom of the atom i. The indices of the neighboring atoms (i.e., atoms within a certain cut-off radius) of atom i are given by the notation $n(i)$. Note that the Cartesian coordinates can be either under the periodic boundary condition (PBC) or in vacuum (under the open boundary condition). The network parameters are denoted by $\boldsymbol{\theta} = \{\boldsymbol{\theta}_d, \boldsymbol{\theta}_f\}$, where $\boldsymbol{\theta}_d$ and $\boldsymbol{\theta}_f$ yield the network parameters of the descriptor (if any)

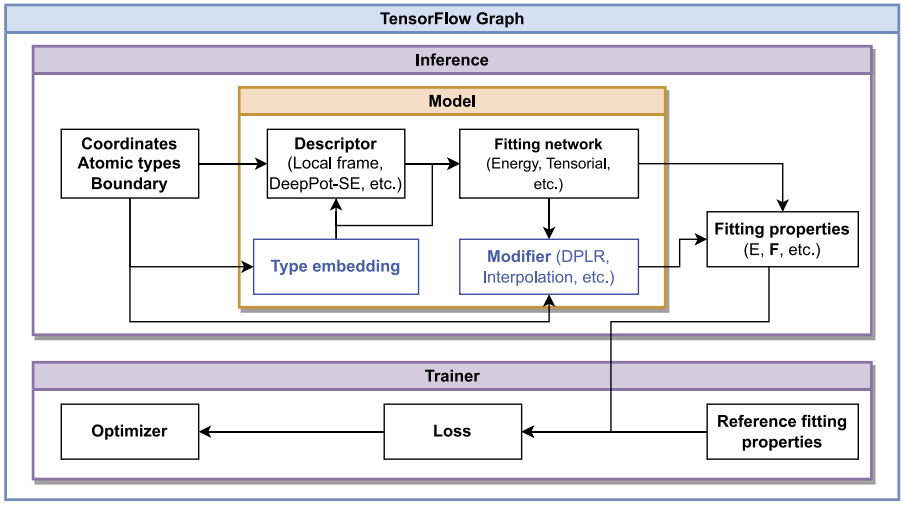

FIG. 1. The components of the DeePMD-kit package. The direction of the arrow indicates the dependency between the components. The blue box represents an optional component.

and those of the fitting network, respectively. From Eq. (1), one may compute the global property of the system by

$$\boldsymbol{y} = \sum_{i=1}^{N} \boldsymbol{y}_i, \tag{2}$$

where N is the number of atoms in a frame. For example, if y_i represents the potential energy contribution of atom i, then y gives the total potential energy of the frame. In the following text, N_c is the expected maximum number of neighboring atoms, which is the same constant for all atoms over all frames. A matrix with a dimension of N_c will be padded if the number of neighboring atoms is less than N_c.

1. Neural networks

A neural network (NN) function $\mathcal{N}$ is the composition of multiple layers $\mathcal{L}^{(i)}$,

$$\mathcal{N} = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)}. \tag{3}$$

In the DeePMD-kit package, a layer $\mathcal{L}$ may be one of the following forms depending on whether a ResNet[111] is used and the number of nodes:

$$\boldsymbol{y} = \mathcal{L}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{b}) = \begin{cases} \hat{\boldsymbol{w}} \odot \boldsymbol{\phi}(\boldsymbol{x}^T \boldsymbol{w} + \boldsymbol{b}) + \boldsymbol{x}, & \text{ResNet and } N_2 = N_1, \\ \hat{\boldsymbol{w}} \odot \boldsymbol{\phi}(\boldsymbol{x}^T \boldsymbol{w} + \boldsymbol{b}) + \{\boldsymbol{x}, \boldsymbol{x}\}, & \text{ResNet and } N_2 = 2N_1, \\ \hat{\boldsymbol{w}} \odot \boldsymbol{\phi}(\boldsymbol{x}^T \boldsymbol{w} + \boldsymbol{b}), & \text{otherwise}, \end{cases} \tag{4}$$

where $\boldsymbol{x} \in \mathbb{R}^{N_1}$ is the input vector and $\boldsymbol{y} \in \mathbb{R}^{N_2}$ is the output vector. $\boldsymbol{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\boldsymbol{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable. $\hat{\boldsymbol{w}} \in \mathbb{R}^{N_2}$ can be either a trainable vector, which represents the "timestep" in the skip connection, or a vector of all ones $\boldsymbol{1} = \{1, 1, \ldots, 1\}$, which disables the time step. $\boldsymbol{\phi}$ is the activation function. In theory, the activation function can be any form, and the following functions are provided in the DeePMD-kit package: hyperbolic tangent (tanh), rectified linear unit (ReLU),[112] ReLU6, softplus,[113] sigmoid, Gaussian error linear unit (GELU),[114] and identity. Among these activation functions, ReLU and ReLU6 are not continuous in the first-order derivative, and others are continuous up to the second-order derivative.

2. Descriptors

DeePMD-kit supports multiple atomic descriptors, including the local frame descriptor, the two-body and three-body embedding DeepPot-SE descriptor, the attention-based descriptor, and the hybrid descriptor that is defined as a combination of multiple descriptors. In the following text, we use $\mathcal{D}^i = \mathcal{D}(\boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j \in n(i)}; \boldsymbol{\theta}_d)$ to represent the atomic descriptor of the atom i.

a. Local frame. The local frame descriptor $\mathcal{D}^i \in \mathbb{R}^{N_c \times \{1,4\}}$ (sometimes simply called the DPMD model), which is the first version of the DP descriptor,[9] is constructed by using either full information or radial-only information,

$$(\mathcal{D}^i)_j = \begin{cases} \left\{ \frac{1}{r_{ij}} \quad \frac{x_{ij}}{r_{ij}} \quad \frac{y_{ij}}{r_{ij}} \quad \frac{z_{ij}}{r_{ij}} \right\}, & \text{full}, \\ \left\{ \frac{1}{r_{ij}} \right\}, & \text{radial} - \text{only}, \end{cases} \tag{5}$$

where (x_{ij}, y_{ij}, z_{ij}) are three Cartesian coordinates of the relative position between atoms i and j, i.e., $\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j = (x_{ij}, y_{ij}, z_{ij})$ in the local frame, and $r_{ij} = |\boldsymbol{r}_{ij}|$ is its norm. In Eq. (5), the order of the neighbors j is sorted in ascending order according to their distance to the atom i. $\boldsymbol{r}_{ij}$ is transformed from the global relative coordinate $\boldsymbol{r}_{ij}^0$ through

$$\boldsymbol{r}_{ij} = \boldsymbol{r}_{ij}^0 \cdot R_i, \tag{6}$$

where

$$R_i = \{\mathbf{e}_{i1}, \mathbf{e}_{i2}, \mathbf{e}_{i3}\} \tag{7}$$

is the rotation matrix constructed by

$$\boldsymbol{e}_{i1} = \boldsymbol{e}(\boldsymbol{r}_{i,a(i)}), \tag{8}$$

$$\boldsymbol{e}_{i2} = \boldsymbol{e}(\boldsymbol{r}_{i,b(i)} - (\boldsymbol{r}_{i,b(i)} \cdot \boldsymbol{e}_{i1})\boldsymbol{e}_{i1}), \tag{9}$$

$$\boldsymbol{e}_{i3} = \boldsymbol{e}_{i1} \times \boldsymbol{e}_{i2}, \tag{10}$$

where $\boldsymbol{e}(\boldsymbol{r}_{ij}) = \boldsymbol{r}_{ij}/r_{ij}$ denotes the operation of normalizing a vector. $a(i) \in n(i)$ and $b(i) \in n(i)$ are the two axis atoms used to define the axes of the local frame of atom i, which, in general, are the two closest atoms, independently of their species, together with the center atom i.

The limitation of the local frame descriptor is that it is not smooth at the cutoff radius and the exchanging of the order of two nearest neighbors [i.e., the swapping of $a(i)$ and $b(i)$], so its usage is limited. We note that the local frame descriptor is the only non-smooth descriptor among all DP descriptors, and we recommend using other descriptors for the usual system.

b. Two-body embedding DeepPot-SE. The two-body embedding smooth edition of the DP descriptor $\mathcal{D}^i \in \mathbb{R}^{M \times M_<}$ is usually named DeepPot-SE descriptor.[10] It is noted that the descriptor is a multi-body representation of the local environment of the atom i. We call it "two-body embedding" because the embedding network takes only the distance between atoms i and j (see below), but it is not implied that the descriptor takes only the pairwise information between i and its neighbors. The descriptor, using either full information or radial-only information, is given by

$$\mathcal{D}^i = \begin{cases} \frac{1}{N_c^2}(\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}^i_<, & \text{full}, \\ \frac{1}{N_c}\sum_j (\mathcal{G}^i)_{jk}, & \text{radial} - \text{only}, \end{cases} \tag{11}$$

where $\mathcal{R}^i \in \mathbb{R}^{N_c \times \{1,4\}}$ is the coordinate matrix, and each row of $\mathcal{R}^i$ can be constructed as

$$(\mathcal{R}^i)_j = \begin{cases} \left\{ s(r_{ij}) \quad \frac{s(r_{ij})x_{ij}}{r_{ij}} \quad \frac{s(r_{ij})y_{ij}}{r_{ij}} \quad \frac{s(r_{ij})z_{ij}}{r_{ij}} \right\}, & \text{full}, \\ \{s(r_{ij})\}, & \text{radial} - \text{only}, \end{cases} \tag{12}$$

where $\boldsymbol{r}_{ij} = \boldsymbol{r}_j - \boldsymbol{r}_i = (x_{ij}, y_{ij}, z_{ij})$ is the relative coordinate and $r_{ij} = \|\boldsymbol{r}_{ij}\|$ is its norm. The switching function $s(r)$ is defined as

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s, \\ \frac{1}{r}\left[x^3(-6x^2 + 15x - 10) + 1\right], & r_s \le r < r_c, \\ 0, & r \ge r_c, \end{cases} \tag{13}$$

where $x = \frac{r - r_s}{r_c - r_s}$ switches from 0 at r_s to 1 at the cutoff radius r_c and $\left[x^3(-6x^2 + 15x - 10) + 1\right]$ switches from 1 at r_s to 0 at r_c. The switching function $s(r)$ is smooth in the sense that the second-order derivative is continuous. The derivation process of the fifth-order polynomial $\left[x^3(-6x^2 + 15x - 10) + 1\right]$ can be found in Appendix A.

Each row of the embedding matrix $\mathcal{G}^i \in \mathbb{R}^{N_c \times M}$ consists of M nodes from the output layer of an NN function $\mathcal{N}_{e,2}$ of $s(r_{ij})$,

$$(\mathcal{G}^i)_j = \mathcal{N}_{e,2}(s(r_{ij})), \tag{14}$$

where the NN function $\mathcal{N}$ was given in Eq. (4) and the subscript "$e,2$" is used to distinguish the NN from other NNs used in the DP model. In Eq. (14), the network parameters are not explicitly written. $\mathcal{G}^i_< \in \mathbb{R}^{N_c \times M_<}$ only takes first $M_<$ columns of $\mathcal{G}^i$ to reduce the size of $\mathcal{D}^i$. r_s, r_c, M, and $M_<$ are hyperparameters provided by the user. Compared to the local frame descriptor, the DeepPot-SE is continuous up to the second-order derivative in its domain.

c. Three-body embedding DeepPot-SE. The three-body embedding DeepPot-SE descriptor incorporates bond-angle information, making the model more accurate.[50] The descriptor $\mathcal{D}^i$ can be represented as

$$\mathcal{D}^i = \frac{1}{N_c^2}(\mathcal{R}^i(\mathcal{R}^i)^T) : \mathcal{G}^i, \tag{15}$$

where $\mathcal{R}^i$ is defined by Eq. (12). Currently, only the full information case of $\mathcal{R}^i$ is supported by the three-body embedding. Similar to Eq. (14), each element of $\mathcal{G}^i \in \mathbb{R}^{N_c \times N_c \times M}$ comes from M nodes from the output layer of an NN $\mathcal{N}_{e,3}$ function,

$$(\mathcal{G}^i)_{jk} = \mathcal{N}_{e,3}((\theta_i)_{jk}), \tag{16}$$

where $(\theta_i)_{jk} = (\mathcal{R}^i)_j \cdot (\mathcal{R}^i)_k$ considers the angle form of two neighbors (j and k). The notation ":" in Eq. (15) indicates the contraction between matrix $\mathcal{R}^i(\mathcal{R}^i)^T$ and the first two dimensions of tensor $\mathcal{G}^i$. The network parameters are also not explicitly written in Eq. (16).

d. Handling the systems composed of multiple chemical species. For a system with multiple chemical species ($|\{\alpha_i\}| > 1$), parameters of the embedding network $\mathcal{N}_{e,\{2,3\}}$ are as follows chemical-species-wise in Eqs. (14) and (16):

$$(\mathcal{G}^i)_j = \mathcal{N}_{e,2}^{\alpha_i,\alpha_j}(s(r_{ij})) \quad \text{or} \quad (\mathcal{G}^i)_j = \mathcal{N}_{e,2}^{\alpha_j}(s(r_{ij})), \tag{17}$$

$$(\mathcal{G}^i)_{jk} = \mathcal{N}_{e,3}^{\alpha_j,\alpha_k}((\theta_i)_{jk}). \tag{18}$$

Thus, there will be N_t^2 or N_t embedding networks, where N_t is the number of chemical species. To improve the performance of matrix operations, $n(i)$ is divided into blocks of different chemical species. Each matrix with a dimension of N_c is divided into corresponding blocks, and each block is padded to $N_c^{\alpha_j}$ separately. The limitation of this approach is that when there are large numbers of chemical species, such as 57 elements in the OC2M dataset,[115,116] the number of embedding networks will become 3249 or 57, requiring large memory and decreasing computing efficiency.

e. Type embedding. To reduce the number of NN parameters and improve computing efficiency when there are large numbers of chemical species, the type embedding $\mathcal{A}$ is introduced, represented as a NN function $\mathcal{N}_t$ of the atomic type α,

$$\mathcal{A}^i = \mathcal{N}_t(\text{one\_hot}(\alpha_i)), \tag{19}$$

where α_i is converted to a one-hot vector representing the chemical species before feeding to the NN. The NN function $\mathcal{N}$ was given in Eq. (4). Based on Eqs. (14) and (16), the type embeddings of central and neighboring atoms $\mathcal{A}^i$ and $\mathcal{A}^j$ are added as an extra input of the embedding network $\mathcal{N}_{e,\{2,3\}}$,

$$(\mathcal{G}^i)_j = \mathcal{N}_{e,2}(\{s(r_{ij}), \mathcal{A}^i, \mathcal{A}^j\}) \quad \text{or} \quad (\mathcal{G}^i)_j = \mathcal{N}_{e,2}(\{s(r_{ij}), \mathcal{A}^j\}), \tag{20}$$

$$(\mathcal{G}^i)_{jk} = \mathcal{N}_{e,3}(\{(\theta_i)_{jk}, \mathcal{A}^j, \mathcal{A}^k\}). \tag{21}$$

In this way, all chemical species share the same network parameters through the type embedding.

f. Attention-based descriptor. An attention-based descriptor $\mathcal{D}^i \in \mathbb{R}^{M \times M_<}$, which is proposed in the pretrainable DPA-1[51] model, is given by

$$\mathcal{D}^i = \frac{1}{N_c^2} (\hat{\mathcal{G}}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \hat{\mathcal{G}}^i_<, \tag{22}$$

where $\hat{\mathcal{G}}^i$ represents the embedding matrix $\mathcal{G}^i$ after additional self-attention mechanism[119] and $\mathcal{R}^i$ is defined by the full case in Eq. (12). Note that we obtain $\mathcal{G}^i$ from Eq. (20) using the type embedding method by default in this descriptor.

To perform the self-attention mechanism, the queries $\mathcal{Q}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, keys $\mathcal{K}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, and values $\mathcal{V}^{i,l} \in \mathbb{R}^{N_c \times d_v}$ are first obtained,

$$\left(\mathcal{Q}^{i,l}\right)_j = Q_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right), \tag{23}$$

$$\left(\mathcal{K}^{i,l}\right)_j = K_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right), \tag{24}$$

$$\left(\mathcal{V}^{i,l}\right)_j = V_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right), \tag{25}$$

where Q_l, K_l, V_l represent three trainable linear transformations that output the queries and keys of dimension d_k and values of dimension d_v and l is the index of the attention layer. The input embedding matrix to the attention layers, denoted by $\mathcal{G}^{i,0}$, is chosen as the two-body embedding matrix (14).

Then, the scaled dot-product attention method[119,118] is adopted,

$$A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l}) = \varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right)\mathcal{V}^{i,l}, \tag{26}$$

where $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) \in \mathbb{R}^{N_c \times N_c}$ is attention weights. In the original attention method, one typically has $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right)$, with $\sqrt{d_k}$ being the normalization temperature. This is slightly modified to incorporate the angular information,

$$\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right) \odot \hat{\mathcal{R}}^i (\hat{\mathcal{R}}^i)^T, \tag{27}$$

where $\hat{\mathcal{R}}^i \in \mathbb{R}^{N_c \times 3}$ denotes normalized relative coordinates, $\hat{\mathcal{R}}^i_j = \frac{\boldsymbol{r}_{ij}}{\|\boldsymbol{r}_{ij}\|}$, and $\odot$ means element-wise multiplication.

Then, layer normalization is added in a residual way to finally obtain the self-attention local embedding matrix $\hat{\mathcal{G}}^i = \mathcal{G}^{i,L_a}$ after L_a attention layers,

$$\mathcal{G}^{i,l} = \mathcal{G}^{i,l-1} + \mathrm{LayerNorm}(A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l})). \tag{28}$$

g. Hybrid descriptor. A hybrid descriptor $\mathcal{D}^i_{\mathrm{hyb}}$ concatenates multiple kinds of descriptors into one descriptor,[53]

$$\mathcal{D}^i_{\mathrm{hyb}} = \{\mathcal{D}^i_1 \quad \mathcal{D}^i_2 \quad \cdots \quad \mathcal{D}^i_n\}. \tag{29}$$

The list of descriptors can be different types or the same descriptors with different parameters. This way, one can set the different cutoff radii for different descriptors.

h. Compression. The compression of the DP model uses three techniques, tabulated inference, operator merging, and precise neighbor indexing, to improve the performance of model training and inference when the model parameters are properly trained.[109]

For better performance, the NN inference can be replaced by tabulated function evaluations if the input of the NN is of dimension one. The embedding networks $\mathcal{N}_{e,2}$ defined by (14) and $\mathcal{N}_{e,3}$ defined by (16) are of this type. The idea is to approximate the output of the NN by a piece-wise polynomial fitting. The input domain (a compact domain in $\mathbb{R}$) is divided into L_c equally spaced intervals, in which we apply a fifth-order polynomial $g^l_m(x)$ approximation of the mth output component of the NN function,

$$g^l_m(x) = a^l_m x^5 + b^l_m x^4 + c^l_m x^3 + d^l_m x^2 + e^l_m x + f^l_m, \quad x \in [x_l, x_{l+1}), \tag{30}$$

where $l = 1, 2, \ldots, L_c$ is the index of the intervals, $x_1, \ldots, x_{L_c}, x_{L_c+1}$ are the endpoints of the intervals, and a^l_m, b^l_m, c^l_m, d^l_m, e^l_m, and f^l_m are the fitting parameters. The fitting parameters can be computed by using the following equations:

$$a^l_m = \frac{1}{2\Delta x_l^5}\left[12h_{m,l} - 6(y'_{m,l+1} + y'_{m,l})\Delta x_l + (y''_{m,l+1} - y''_{m,l})\Delta x_l^2\right], \tag{31}$$

$$b^l_m = \frac{1}{2\Delta x_l^4}\left[-30h_{m,l} + (14y'_{m,l+1} + 16y'_{m,l})\Delta x_l + (-2y''_{m,l+1} + 3y''_{m,l})\Delta x_l^2\right], \tag{32}$$

$$c^l_m = \frac{1}{2\Delta x_l^3}\left[20h_{m,l} - (8y'_{m,l+1} + 12y'_{m,l})\Delta x_l + (y''_{m,l+1} - 3y''_{m,l})\Delta x_l^2\right], \tag{33}$$

$$d^l_m = \frac{1}{2}y''_{m,l}, \tag{34}$$

$$e^l_m = y'_{m,l}, \tag{35}$$

$$f^l_m = y_{m,l}, \tag{36}$$

where $\Delta x_l = x_{l+1} - x_l$ denotes the size of the interval. $h_{m,l} = y_{m,l+1} - y_{m,l}$. $y_{m,l} = y_m(x_l)$, $y'_{m,l} = y'_m(x_l)$, and $y''_{m,l} = y''_m(x_l)$ are the value, the first-order derivative, and the second-order derivative of the mth

component of the target NN function at the interval point x_l, respectively. The first- and second-order derivatives are easily calculated by the back-propagation of the NN functions.

In the standard DP model inference, taking the two-body embedding descriptor as an example, the matrix product $(\mathcal{G}^i)^T\mathcal{R}$ requires the transfer of the tensor $\mathcal{G}^i$ between the register and the host/device memories, which usually becomes the bottle-neck of the computation due to the relatively small memory bandwidth of the GPUs. The compressed DP model merges the matrix multiplication $(\mathcal{G}^i)^T\mathcal{R}$ with the tabulated inference step. More specifically, once one column of $(\mathcal{G}^i)^T$ is evaluated, it is immediately multiplied with one row of the environment matrix in the register, and the outer product is deposited to the result of $(\mathcal{G}^i)^T\mathcal{R}$. By the operator merging technique, the allocation of $\mathcal{G}^i$ and the memory movement between register and host/device memories is avoided. The operator merging of the three-body embedding can be derived analogously.

The first dimension, N_c, of the environment $(\mathcal{R}^i)$ and embedding $(\mathcal{G}^i)$ matrices is the expected maximum number of neighbors. If the number of neighbors of an atom is smaller than N_c, the corresponding positions of the matrices are pad with zeros. In practice, if the real number of neighbors is significantly smaller than N_c, a notable operation is spent on the multiplication of padding zeros. In the compressed DP model, the number of neighbors is precisely indexed at the tabulated inference stage, further saving computational costs.

3. Fitting networks

The fitting network can fit the potential energy of a system, along with the force and the virial, and tensorial properties, such as the dipole and the polarizability.

a. Fitting potential energies. In the DP model (1), we let the fitting network $\mathcal{F}_0$ map the descriptor $\mathcal{D}^i$ to a scalar, where the subscript "0" means that the output is a zero-order tensor (i.e., scalar). The model can then be used to predict the total potential energy of the system by

$$E = \sum_i E_i = \sum_i \mathcal{F}_0(\mathcal{D}^i), \tag{37}$$

where the output of the fitting network is treated as the atomic potential energy contribution, i.e., E_i. The output scalar can also be treated as other scalar properties defined on an atom, for example, the partial charge of atom i.

In some cases, atomic-specific or frame-specific parameters, such as electron temperature,[119] may be treated as extra input to the fitting network. We denote the atomic and frame-specific parameters by $\boldsymbol{P}^i \in \mathbb{R}^{N_p}$ (with N_p being the dimension) and $\boldsymbol{Q} \in \mathbb{R}^{N_q}$ (with N_q being the dimension), respectively,

$$E_i = \mathcal{F}_0(\{\mathcal{D}^i, \boldsymbol{P}^i, \boldsymbol{Q}\}). \tag{38}$$

The atomic force $\boldsymbol{F}_i$ and the virial tensor $\Xi = (\Xi_{\alpha\beta})$ (if PBC is applied) can be derived from the potential energy E,

$$F_{i,\alpha} = -\frac{\partial E}{\partial r_{i,\alpha}}, \tag{39}$$

$$\Xi_{\alpha\beta} = -\sum_\gamma \frac{\partial E}{\partial h_{\gamma\alpha}} h_{\gamma\beta}, \tag{40}$$

where $r_{i,\alpha}$ and $F_{i,\alpha}$ denote the αth component of the coordinate and force of atom i. $h_{\alpha\beta}$ is the βth component of the αth basis vector of the simulation region.

b. Fitting tensorial properties. To represent the first-order tensorial properties (i.e., vector properties), we let the fitting network, denoted by $\mathcal{F}_1$, output an M-dimensional vector; then, we have the representation

$$(T_i^{(1)})_\alpha = \frac{1}{N_c}\sum_{j=1}^{N_c}\sum_{m=1}^{M} (\mathcal{G}^i)_{jm}(\mathcal{R}^i)_{j,\alpha+1}(\mathcal{F}_1(\mathcal{D}^i))_m, \ \alpha = 1,2,3. \tag{41}$$

We let the fitting network $\mathcal{F}_2$ output an M-dimensional vector, and the second-order tensorial properties (matrix properties) are formulated as

$$(T_i^{(2)})_{\alpha\beta} = \frac{1}{N_c^2}\sum_{j=1}^{N_c}\sum_{k=1}^{N_c}\sum_{m=1}^{M} (\mathcal{G}^i)_{jm}(\mathcal{R}^i)_{j,\alpha+1}(\mathcal{R}^i)_{k,\beta+1} \times (\mathcal{G}^i)_{km}(\mathcal{F}_2(\mathcal{D}^i))_m, \quad \alpha,\beta = 1,2,3, \tag{42}$$

where $\mathcal{G}^i$ and $\mathcal{R}^i$ can be found at Eqs. (12) and (14) (full case), respectively. Thus, the tensor fitting network requires the descriptor to have the same or similar form as the DeepPot-SE descriptor. The NN functions $\mathcal{F}_1$ and $\mathcal{F}_2$ were given in Eq. (4). The total tensor $\boldsymbol{T}$ (total dipole $\boldsymbol{T}^{(1)}$ or total polarizability $\boldsymbol{T}^{(2)}$) is the sum of the atomic tensor,

$$\boldsymbol{T} = \sum_i \boldsymbol{T}_i. \tag{43}$$

The tensorial models can be used to calculate the IR spectrum[105] and Raman spectrum.[106]

c. Handling the systems composed of multiple chemical species. Similar to the embedding networks, if the type embedding approach is not used, the fitting network parameters are chemical-species-wise, and there are N_t sets of fitting network parameters. For performance, atoms are sorted by their chemical species α_i in advance. Taking an example, the atomic energy E_i is represented as follows based on Eq. (38):

$$E_i = \mathcal{F}_0^{\alpha_i}(\mathcal{D}^i). \tag{44}$$

When the type embedding is used, all chemical species share the same network parameters, and the type embedding is inserted into the input of the fitting networks in Eq. (38),

$$E_i = \mathcal{F}_0(\{\mathcal{D}^i, \mathcal{A}^i\}). \tag{45}$$

4. Deep potential range correction (DPRc)

Deep Potential-Range Correction (DPRc)[52,75] was initially designed to correct the potential energy from a fast, linear-scaling low-level semiempirical QM/MM theory to a high-level *ab initio*

QM/MM theory in a range-correction way to quantitatively correct short and mid-range non-bonded interactions leveraging the non-bonded lists routinely used in molecular dynamics simulations using molecular mechanical force fields, such as AMBER.[120] In this way, long-ranged electrostatic interactions can be modeled efficiently using the particle mesh Ewald method[120] or its extensions for multipolar[94,95] and QM/MM[96,97] potentials. In a DPRc model, the switch function in Eq. (13) is modified to disable MM–MM interaction,

$$s_{\mathrm{DPRc}}(r_{ij}) = \begin{cases} 0 & \text{if } i \in \mathrm{MM} \wedge j \in \mathrm{MM}, \\ s(r_{ij}) & \text{otherwise}, \end{cases} \tag{46}$$

where $s_{\mathrm{DPRc}}(r_{ij})$ is the new switch function and $s(r_{ij})$ is the old one in Eq. (13). This ensures that the forces between MM atoms are zero, i.e.,

$$\boldsymbol{F}_{ij} = -\frac{\partial E}{\partial \boldsymbol{r}_{ij}} = 0, \quad i \in \mathrm{MM} \wedge j \in \mathrm{MM}. \tag{47}$$

The fitting network in Eq. (38) is revised to remove energy bias from MM atoms,

$$E_i = \begin{cases} \mathcal{F}_0(\mathcal{D}^i) & \text{if } i \in \mathrm{QM}, \\ \mathcal{F}_0(\mathcal{D}^i) - \mathcal{F}_0(\mathbf{0}) & \text{if } i \in \mathrm{MM}, \end{cases} \tag{48}$$

where $\mathbf{0}$ is a zero matrix. It is worth mentioning that the usage of DPRc is not limited to its initial design for QM/MM correction and can be expanded to any similar interaction.[121]

5. Deep potential long range (DPLR)

The Deep Potential Long Range (DPLR) model adds the electrostatic energy to the total energy,[53]

$$E = E_{\mathrm{DP}} + E_{\mathrm{ele}}, \tag{49}$$

where E_{DP} is the short-range contribution constructed as the standard energy model in Eq. (37) that is fitted against $(E^* - E_{\mathrm{ele}})$. E_{ele} is the electrostatic energy introduced by a group of Gaussian distributions that is an approximation of the electronic structure of the system and is calculated in Fourier space by

$$E_{\mathrm{ele}} = \frac{1}{2\pi V} \sum_{m \neq 0, \|m\| \leq L} \frac{\exp\left(-\pi^2 m^2/\beta^2\right)}{m^2} S^2(m), \tag{50}$$

where β is a freely tunable parameter that controls the spread of the Gaussians. L is the cutoff in Fourier space, and $S(m)$, the structure factor, is given by

$$S(m) = \sum_i q_i e^{-2\pi \imath m \boldsymbol{r}_i} + \sum_n q_n e^{-2\pi \imath m \boldsymbol{W}_n}, \tag{51}$$

where $\imath = \sqrt{-1}$ denotes the imaginary unit, $\boldsymbol{r}_i$ indicates ion coordinates, q_i is the charge of the ion i, and W_n is the nth Wannier centroid (WC), which can be obtained from a separated dipole model in Eq. (42). It can be proved that the error in the electrostatic energy introduced by the Gaussian approximations is dominated by a summation of dipole-quadrupole interactions that decay as r^{-4}, where r is the distance between the dipole and quadrupole.[53]

6. Interpolation with a pairwise potential

In applications such as the radiation damage simulation, the interatomic distance may become too close so that the DFT calculations fail. In such cases, the DP model that is an approximation of the DFT potential energy surface is usually replaced by an empirical potential, such as the Ziegler–Biersack–Littmark (ZBL)[122] screened nuclear repulsion potential in radiation damage simulations.[123] The DeePMD-kit package supports the interpolation between DP and an empirical pairwise potential,

$$E_i = (1 - w_i) E_i^{\mathrm{DP}} + w_i E_i^{\mathrm{pair}}, \tag{52}$$

where w_i is the interpolation weight and E_i^{pair} is the atomic contribution due to the pairwise potential $u^{\mathrm{pair}}(r)$, i.e.,

$$E_i^{\mathrm{pair}} = \sum_{j \in n(i)} u^{\mathrm{pair}}(r_{ij}). \tag{53}$$

The interpolation weight w_i is defined by

$$w_i = \begin{cases} 1, & \sigma_i < r_a, \\ u_i^3(-6u_i^2 + 15u_i - 10) + 1, & r_a \leq \sigma_i < r_b, \\ 0, & \sigma_i \geq r_b, \end{cases} \tag{54}$$

where $u_i = (\sigma_i - r_a)/(r_b - r_a)$. The derivation process of Eq. (54) can be found in Appendix A. In the range $[r_a, r_b]$, the DP model smoothly switched off and the pairwise potential smoothly switched on from r_b to r_a. σ_i is the softmin of the distance between atom i and its neighbors,

$$\sigma_i = \frac{\sum_{j \in n(i)} r_{ij} e^{-r_{ij}/\alpha_s}}{\sum_{j \in n(i)} e^{-r_{ij}/\alpha_s}}, \tag{55}$$

where the scale α_s is a tunable scale of the interatomic distance r_{ij}. The pairwise potential $u^{\mathrm{pair}}(r)$ is defined by a user-defined table that provides the value of u^{pair} on an evenly discretized grid from 0 to the cutoff distance.

B. Trainer

Based on DP models $\mathcal{M}$ defined in Eq. (1), a trainer should also be defined to train parameters in the model, including weights and biases in Eq. (4). The learning rate γ, the loss function L, and the training process should be given in a trainer.

1. Learning rate

The learning rate γ decays exponentially,

$$\gamma(\tau) = \gamma^0 r^{\lfloor \tau/s \rfloor}, \tag{56}$$

where $\tau \in \mathbb{N}$ is the index of the training step, $\gamma^0 \in \mathbb{R}$ is the learning rate at the first step, and the decay rate r is given by

$$r = \left(\frac{\gamma^{\mathrm{stop}}}{\gamma^0}\right)^{\frac{s}{\tau^{\mathrm{stop}}}}, \tag{57}$$

where $\tau^{\text{stop}} \in \mathbb{N}$, $\gamma^{\text{stop}} \in \mathbb{R}$, and $s \in \mathbb{N}$ are the stopping step, the stopping learning rate, and the decay steps, respectively, all of which are hyperparameters provided in advance.

2. Loss function

The loss function L is given by a weighted sum of different fitting property loss L_p,

$$L(\boldsymbol{x};\boldsymbol{\theta},\tau) = \frac{1}{\mathcal{B}}\sum_{k\in\mathcal{B}}\sum_{\eta} p_\eta(\tau)L_\eta(\boldsymbol{x}^k;\boldsymbol{\theta}), \tag{58}$$

where $\mathcal{B}$ is the mini-batch of data. $\boldsymbol{x} = \{\boldsymbol{x}^k\}$ is the dataset. $\boldsymbol{x}^k = (\boldsymbol{x}_1^k, \ldots, \boldsymbol{x}_N^k)$ is a single data frame from the set and is composed of all the degrees of freedom of the atoms. η denotes the property to be fit. For each property, p_η is a prefactor given by

$$p_\eta(\tau) = p_\eta^{\text{limit}}\left(1 - \frac{\gamma(\tau)}{\gamma^0}\right) + p_\eta^{\text{start}}\frac{\gamma(\tau)}{\gamma^0}, \tag{59}$$

where p_η^{start} and p_η^{limit} are hyperparameters that give the prefactor at the first training step and the infinite training steps, respectively. $\gamma(\tau)$ is the learning rate defined by Eq. (56).

The loss function of a specific fitting property L_η is defined by the mean squared error (MSE) of a data frame and is normalized by the number of atoms N if η is a frame property that is a linear combination of atomic properties. Taking an example, if an energy model is fitted as given in Eq. (37), the properties η could be energy E, force $\boldsymbol{F}$, virial Ξ, relative energy ΔE,[72] or any combination among them, and the loss functions of them are

$$L_E(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{N}(E(\boldsymbol{x};\boldsymbol{\theta}) - E^*)^2, \tag{60}$$

$$L_F(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{3N}\sum_{k=1}^{N}\sum_{\alpha=1}^{3}(F_{k,\alpha}(\boldsymbol{x};\boldsymbol{\theta}) - F_{k,\alpha}^*)^2, \tag{61}$$

$$L_\Xi(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{9N}\sum_{\alpha,\beta=1}^{3}(\Xi_{\alpha\beta}(\boldsymbol{x};\boldsymbol{\theta}) - \Xi_{\alpha\beta}^*)^2, \tag{62}$$

$$L_{\Delta E}(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{N}(\Delta E(\boldsymbol{x};\boldsymbol{\theta}) - \Delta E^*)^2, \tag{63}$$

where $F_{k,\alpha}$ is the αth component of the force on atom k and the superscript "*" indicates the label of the property that should be provided in advance. Using N ensures that each loss of fitting property is averaged over atomic contributions before they contribute to the total loss by weight.

If part of atoms is more important than others, for example, certain atoms play an essential role when calculating free energy profiles or kinetic isotope effects,[52,75] the MSE of atomic forces with prefactors q_k can also be used as the loss function,

$$L_F^p(\mathbf{x};\boldsymbol{\theta}) = \frac{1}{3N}\sum_{k=1}^{N}\sum_{\alpha} q_k(F_{k,\alpha}(\mathbf{x};\boldsymbol{\theta}) - F_{k,\alpha}^*)^2. \tag{64}$$

The atomic forces with larger prefactors will be fitted more accurately than those in other atoms.

If some forces are quite large, for example, forces can be greater than 60 eV/Å in high-temperature reactive simulations,[77,78] one may also prefer that the force loss is relative to the magnitude instead of Eq. (61),

$$L_F^r(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{3N}\sum_{k=1}^{N}\sum_{\alpha}\left(\frac{F_{k,\alpha}(\boldsymbol{x};\boldsymbol{\theta}) - F_{k,\alpha}^*}{|\boldsymbol{F}_k^*| + \nu}\right)^2, \tag{65}$$

where ν is a small constant used to protect an atom where the magnitude of $\boldsymbol{F}_k^*$ is small from having a large L_F^r. Benefiting from the relative force loss, small forces can be fitted more accurately.

3. Training process

During the training process, the loss function is minimized by the stochastic gradient descent algorithm Adam.[124] Ideally, the resulting parameter is the minimizer of the loss function,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \lim_{\tau\to+\infty} L(\boldsymbol{x};\boldsymbol{\theta},\tau). \tag{66}$$

In practice, the Adam optimizer stops at the step τ_{stop}, and the learning rate varies according to scheme (56). τ_{stop} is a hyperparameter usually set to several million.

4. Multiple task training

The multi-task training process can simultaneously handle different datasets with properties that cannot be fitted in one network (e.g., properties from DFT calculations under different exchange–correlation functionals or different basis sets). These datasets are denoted by $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n_t)}$. For each dataset, a training task is defined as

$$\min_{\boldsymbol{\theta}} L^{(t)}(\boldsymbol{x}^{(t)};\boldsymbol{\theta}^{(t)},\tau), \quad t = 1, \ldots, n_t. \tag{67}$$

During the multi-task training process, all tasks share one descriptor with trainable parameters $\boldsymbol{\theta}_d$, while each of them has its own fitting network with trainable parameters $\boldsymbol{\theta}_f^{(t)}$; thus, $\boldsymbol{\theta}^{(t)} = \{\boldsymbol{\theta}_d, \boldsymbol{\theta}_f^{(t)}\}$. At each training step, a task is randomly picked from $1, \ldots, n_t$, and the Adam optimizer is executed to minimize $L^{(t)}$ for one step to update the parameter $\boldsymbol{\theta}^{(t)}$. If different fitting networks have the same architecture, they can share the parameters of some layers to improve training efficiency.

C. Model deviation

Model deviation ϵ_y is the standard deviation of properties $\boldsymbol{y}$ inferred by an ensemble of models $\mathcal{M}_1, \ldots, \mathcal{M}_{n_m}$ that are trained by the same dataset(s) with the model parameters initialized independently. The DeePMD-kit supports $\boldsymbol{y}$ to be the atomic force $\boldsymbol{F}_i$ and the virial tensor Ξ. The model deviation is used to estimate the error of a model at a certain data frame, denoted by $\boldsymbol{x}$, containing the coordinates and chemical species of all atoms. We present the model deviation of the atomic force and the virial tensor,

$$\epsilon_{\boldsymbol{F},i}(\boldsymbol{x}) = \sqrt{\langle\|\boldsymbol{F}_i(\boldsymbol{x};\boldsymbol{\theta}_k) - \langle\boldsymbol{F}_i(\boldsymbol{x};\boldsymbol{\theta}_k)\rangle\|^2\rangle}, \tag{68}$$

$$\epsilon_{\Xi,\alpha\beta}(\boldsymbol{x}) = \frac{1}{N}\sqrt{\langle(\Xi_{\alpha\beta}(\boldsymbol{x};\boldsymbol{\theta}_k) - \langle\Xi_{\alpha\beta}(\boldsymbol{x};\boldsymbol{\theta}_k)\rangle)^2\rangle}, \tag{69}$$

where $\boldsymbol{\theta}_k$ is the parameter of the model $\mathcal{M}_k$ and the ensemble average $\langle\cdot\rangle$ is estimated by

$$\langle \boldsymbol{y}(\boldsymbol{x};\boldsymbol{\theta}_k)\rangle = \frac{1}{n_m}\sum_{k=1}^{n_m} \boldsymbol{y}(\boldsymbol{x};\boldsymbol{\theta}_k). \tag{70}$$

Small $\epsilon_{\boldsymbol{F},i}$ means the model has learned the given data; otherwise, it is not covered, and the training data needs to be expanded. If the magnitude of $\boldsymbol{F}_i$ or Ξ is quite large, a relative model deviation $\epsilon_{\boldsymbol{F},i,\mathrm{rel}}$ or $\epsilon_{\Xi,\alpha\beta,\mathrm{rel}}$ can be used instead of the absolute model deviation,[78]

$$\epsilon_{\boldsymbol{F},i,\mathrm{rel}}(\boldsymbol{x}) = \frac{|\epsilon_{\boldsymbol{F},i}(\boldsymbol{x})|}{|\langle \boldsymbol{F}_i(\boldsymbol{x};\boldsymbol{\theta}_k)\rangle| + \nu}, \tag{71}$$

$$\epsilon_{\Xi,\alpha\beta,\mathrm{rel}}(\boldsymbol{x}) = \frac{\epsilon_{\Xi,\alpha\beta}(\boldsymbol{x})}{|\langle \Xi(\boldsymbol{x};\boldsymbol{\theta}_k)\rangle| + \nu}, \tag{72}$$

where ν is a small constant used to protect an atom where the magnitude of $\boldsymbol{F}_i$ or Ξ is small from having a large model deviation.

Statistics of $\epsilon_{\boldsymbol{F},i}$ and $\epsilon_{\Xi,\alpha\beta}$ can be provided, including the maximum, average, and minimal model deviation over the atom index i and over the component index α,β, respectively. The maximum model deviation of forces $\epsilon_{\boldsymbol{F},\max}$ in a frame was found to be the best error indicator in a concurrent or active learning algorithm.[103,107]

III. TECHNICAL IMPLEMENTATION

In addition to incorporating new powerful features, DeePMD-kit has been designed with the following goals in mind: high performance, high usability, high extensibility, and community engagement. These goals are crucial for DeePMD-kit to become a widely-used platform across various computational fields. In this section, we will introduce several technical implementations that have been put in place to achieve these goals.

A. Code architecture

The DeePMD-kit utilizes TensorFlow's computational graph architecture to construct its DP models,[125] which are composed of various operators implemented with C++, including customized ones, such as the environment matrix, Ewald summation, compressed operator, and their backward propagations. The autograd mechanism provided by TensorFlow is used to compute the derivatives of the DP model with respect to the input atomic coordinates and simulation cell tensors. To optimize performance, some of the critical customized operators are implemented for GPU execution using CUDA or ROCm toolkit libraries. The DeePMD-kit provides Python, C++, and C APIs for inference, facilitating easy integration with third-party software packages. As indicated in Fig. 2, the code of the DeePMD-kit consists of the following modules:

- The core C++ library provides the implementation of customized operators, such as the atomic environmental matrix, neighbor lists, and compressed neural networks. It is important to note that the core C++ library is independently built and tested without TensorFlow's C++ interface.
- The GPU library (CUDA[126] or ROCm[127]), an optional part of the core C++ library, is used to compute customized operators on GPU devices other than central processing units (CPUs). This library depends on the GPU toolkit library (NVIDIA CUDA Toolkit or AMD ROCm Toolkit) and is also independently built and tested.
- The DP operators library contains several customized operators not supported by TensorFlow.[125] TensorFlow provides both Python and C++ interfaces to implement some customized operators, with the TensorFlow C++ library packaged inside its Python package.
- The "model definition" module, written in Python, is used to generate computing graphs composed of TensorFlow operators, DP customized operators, and model parameters organized as "variables." The graph can be saved into a file that can be restored for inference. It depends on the TensorFlow Python API (version 1, `tf.compat.v1`) and other Python dependencies, such as the NumPy[128] and H5Py[129] packages.
- The Python application programming interface (API) is used for inference and can read computing graphs from a file and use the TensorFlow Python API to execute the graph.
- The C++ API, built on the TensorFlow C++ interface, does the same thing as the Python API for inference.
- The C API is a wrapper of the C++ API and provides the same features as the C++ API. Compared to the C++ API, the C API has a more stable application binary interface (ABI) and ensures backward compatibility.
- The header-only C++ API is a wrapper of the C API and provides the same interface as the C++ API. It has the same stable ABI as the C API but still takes advantage of the flexibility of C++.
- The command line interface (CLI) is provided to both general users and developers and is used for both training and inference. It depends on the model definition module and the Python API.

The CMake build system[130] manages all modules, and the pip and scikit-build[131] packages are used to distribute DeePMD-kit as a Python package. The standard Python unit testing framework[132] is used for unit tests on all Python codes, while GoogleTest software[133] is used for tests on all C++ codes. GitHub Actions automates build, test, and deployment pipelines.

B. Performance

1. Hardware acceleration

In the TensorFlow framework, a static graph combines multiple operators with inputs and outputs. Two kinds of operators are time-consuming during training or inference. The first one is TensorFlow's native operators for neural networks (see Sec. II A 1) and matrix operations, which have been fully optimized by the TensorFlow framework itself[125] for both CPU and GPU architectures. Second, the DeePMD-kit's customized operators are for computing the atomic environment [Eqs. (6) and (12)], for interpolation with a pairwise potential, and for the tabulated inference of the embedding matrix [Eq. (30)]. These operators are not supported by the TensorFlow framework but can be accelerated using

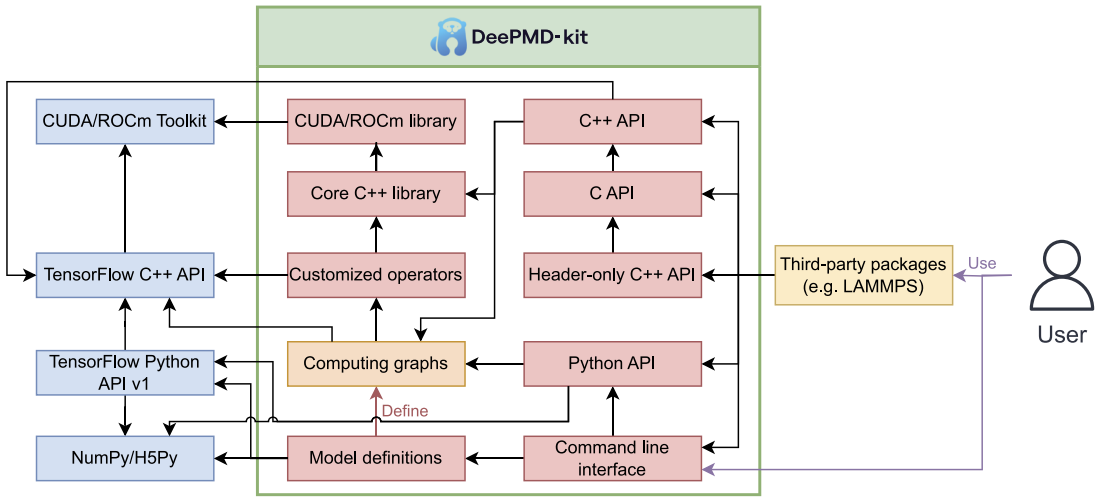

FIG. 2. The architecture of the DeePMD-kit code. The red boxes are modules within the DeePMD-kit package (the green box), the orange box represents computing graphs, the blue boxes are dependencies of the DeePMD-kit, and the yellow box represents third-party packages integrated with DeePMD-kit, including LAMMPS, i-PI, GROMACS, AMBER, OpenMM, ABACUS, ASE, MAGUS, DP-Data, DP-GEN, and MLatom. Customized operators are operators that are not offered by TensorFlow, including atomic environmental matrix, interpolation with a pairwise potential, and tabulated inference of the embedding matrix. The direction of the black arrow A → B indicates that module A is dependent on module B. The red and purple arrows represent "define" and "use," respectively.

OpenMP,[134] CUDA,[126] and ROCm[127] for parallelization under both CPUs and GPUs, except the features without GPU supports listed in Appendix B.

The operator of the environment matrix includes two steps:[108] formatting the neighbor list and computing the matrix elements of $\mathcal{R}$. In the formatting step, the neighbors of the atom i are sorted according to their type α_j, their distance r_{ij} to atom i, and finally their index j. To improve sorting performance on GPUs, the atomic type, distance, and index are compressed into a 64-bit integer $S \in \mathbb{N}$ used for sorting,

$$S = \alpha_j \times 10^{15} + \lfloor r_{ij} \times 10^8 \rfloor \times 10^5 + j. \tag{73}$$

The sorted neighbor index is decompressed from the sorted S and then used to format the neighbor list.

2. MPI implementation for multi-device training and MD simulations

Users may prefer to utilize multiple CPU cores, GPUs, or hardware across multiple nodes to achieve faster performance and larger memory during training or molecular dynamics (MD) simulations. To facilitate this, DeePMD-kit has added message-passing interface (MPI) implementation[135,136] for multi-device training and MD simulations in two ways, which are described below.

Multi-device training is conducted with the help of Horovod, a distributed training framework.[137] Horovod works in the data-parallel mode by equally distributing a batch of data among workers along the axis of the batch size $\mathcal{B}$.[138] During training, each worker consumes sliced input records at different offsets, and only the trainable parameter gradients are averaged with peers. This design avoids batch size and tensor shape conflicts and reduces the number of bytes that need to be communicated among processes. The mpi4py package[139] is used to remove redundant logs.

Multi-device MD simulations are implemented by utilizing the existing parallelism features of third-party MD packages. For example, a Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) enables parallelism across CPUs by optimizing partitioning, communication, and neighbor lists.[140] AMBER builds a similar neighbor list in the interface to DeePMD-kit.[52,54,141] DeePMD-kit supports local atomic environment calculation and accepts the neighbor list $n(i)$ from other software to replace the native neighbor list calculation.[108] In a device, the neighbors from other devices are considered "ghost" atoms that do not contribute atomic energy E_i to this device's total potential energy E.

3. Non-von Neumann molecular dynamics (NVNMD)

When performing molecular dynamics (MD) simulations on CPUs and GPUs, a large majority of time and energy (e.g., more than 95%) is consumed by the DP model inference. This inference process is limited by the "memory wall" and "power wall" bottlenecks of von Neumann (vN) architecture, which means that a significant amount of time and energy (e.g., over 90%) is wasted on data transfer between the processor and memory. As a result, it is difficult to improve computational efficiency.

To address these challenges, non-von Neumann molecular dynamics (NVNMD) uses a non-von Neumann (NvN) architecture chip to accelerate inference. The NvN chip contains processing and memory units that can be used to implement the DP algorithm. In the NvN chip, the hardware algorithm runs fully pipelined. The model parameters are stored in on-chip memory after being loaded from off-chip memory during the initialization process. Therefore, two components of data shuttling are avoided: (1) reading/writing the intermediate results from/to off-chip memory and (2) loading model parameters from off-chip memory during the calculation process. As a result, the DP model ensures high accuracy with NVNMD,

while the NvN chip ensures high computational efficiency. For more details, see Ref. 110.

C. Usability

1. Documentation

DeePMD-kit's features and arguments have grown rapidly with more and more development. To address this issue, we have introduced Sphinx[142] and Doxygen[143] to manage and generate documentation for developers from docstrings in the code. We use the DArgs package (see Sec. III E) to automatically generate Sphinx documentation for user input arguments. The documentation is currently hosted on Read the Docs (https://docs.deepmodeling.org/projects/deepmd/). Furthermore, we strive to make the error messages raised by DeePMD-kit clear to users. In addition, the GitHub Discussion forum allows users to ask questions and receive answers. Recently, several tutorials have been published[49,54] to help new users quickly learn DeePMD-kit.

2. Easy installation

As shown in Fig. 2, DeePMD-kit has dependencies on both Python and C++ libraries of TensorFlow, which can make it difficult and time-consuming for new users to build TensorFlow and DeePMD-kit from the source code. Therefore, we provide compiled binary packages that are distributed via pip, Conda (DeepModeling and conda-forge[144] channels), Docker, and offline packages for Linux, macOS, and Windows platforms. With the help of these precompiled binary packages, users can install DeePMD-kit in just a few minutes. These binary packages include DeePMD-kit's LAMMPS plugin, i-PI driver, and GROMACS patch. As LAMMPS provides a plugin mode in its latest version, DeePMD-kit's LAMMPS plugin can be compiled without having to re-compile LAMMPS.[140] We offer a compiled binary package that includes the C API and the header-only C++ API, making it simpler to integrate with sophisticated software, such as AMBER.[52,54,141]

3. User interface

DeePMD-kit offers a command line interface (CLI) for training, freezing, and testing models. In addition to CLI arguments, users must provide a JSON[145] or YAML[146] file with completed arguments for components listed in Sec. II. The DArgs package (see Sec. III E) parses these arguments to check if user input is correct. An example of how to use the user interface is provided in Ref. 54. Users can also use DP-GUI (see Sec. III E) to fill in arguments in an interactive web page and save them to a JSON[145] file.

DeePMD-kit provides an automatic algorithm that assists new users in deciding on several arguments. For example, the automatic batch size $\mathcal{B}$ determines the maximum batch size during training or inferring to fully utilize memory on a GPU card. The automatic neighbor size N_c determines the maximum number of neighbors by stating the training data to reduce model memory usage. The automatic probability determines the probability of using a system during training. These automatic arguments reduce the difficulty of learning and using the DeePMD-kit.

4. Input data

To train and test models, users are required to provide fitting data in a specified format. DeePMD-kit supports two file formats for data input: NumPy binary files[128] and HDF5 files.[147] These formats are designed to offer superior performance when read by the program with parallel algorithms compared to text files. HDF5 files have the advantage of being able to store multiple arrays in a single file, making them easier to transfer between machines. The Python package "DP-Data" (see Sec. III E) can generate these files from the output of an electronic calculation package.

5. Model visualization

DeePMD-kit supports most of the visualization features offered by TensorBoard,[125] such as tracking and visualizing metrics, viewing the model graph, histograms of tensors, summaries of trainable variables, and debugging profiles.

D. Extensibility

1. Application programming interface and third-party software

DeePMD-kit offers various APIs, including the Python, C++, C, and header-only C++ API, as well as a command-line interface (CLI), as shown in Fig. 2. These APIs are primarily used for inference by developers and high-level users in different situations. Sphinx[142] generates the API details in the documentation.

These APIs can be easily accessed by various third-party software. The Python API, for instance, is utilized by third-party Python packages, such as Atomic Simulation Environment (ASE),[148] MAGUS,[149] and DP-Data (see Sec. III E). The C++, C, or header-only C++ API has also been integrated into several third-party MD packages, such as LAMMPS,[140,150] i-PI,[151] GROMACS,[152] AMBER,[52,54,141] OpenMM,[153,154] and ABACUS.[155] Moreover, the CLI is called by various third-party workflow packages, such as DP-GEN[107] and MLatom.[34] While the ASE calculator, the LAMMPS plugin, the i-PI driver, and the GROMACS patch are developed within the DeePMD-kit code, others are distributed separately. By integrating these APIs into their programs, researchers can perform simulations and minimization, without being restricted by DeePMD-kit's software features.[72,75,83,156] Additionally, they can combine DP models with other potentials outside the DeePMD-kit package if necessary.[52,72,157]

Molecular dynamics, a primary application for DP models, is facilitated by several third-party packages that interface with the DeePMD-kit package, offering a wide range of supported features:

- LAMMPS[140] is seamlessly integrated with the DeePMD-kit through a dedicated plugin developed within the DeePMD-kit project. This plugin supports MPI, as discussed in Sec. III B 2, and provides essential functionalities, such as force calculations. Additionally, it enables on-the-fly computation of model deviation, as shown in Eqs. (68)–(72), during concurrent learning. The plugin can obtain atomic and frame-specific parameters in Eq. (38) from various sources, including constants, electronic temperatures calculated by LAMMPS, or any `compute` style from LAMMPS. LAMMPS also supports calculating classical point charges'

long-range (Coulomb) interaction using the Ewald summation and the fast algorithm particle–particle particle–mesh Ewald (PPPM). The k-space part of these methods, involving the Fourier space transformation of Gaussian charge distributions to compute the Coulomb interaction[158] [as shown in Eq. (50)], is utilized by the DPLR method to handle the long-range interaction.

- i-PI[151] is integrated with the DeePMD-kit through a dedicated driver provided within the DeePMD-kit project. The driver enables the path integral molecular dynamics (PIMD) driven by the i-PI engine and is compatible with the MolSSI Driver Interface (MDI) package[159] and a similar interface in the ASE package.[148] However, the communication between the i-PI driver and the engine relies on UNIX-domain sockets or the network interface, which can limit performance. To overcome this limitation, developers have incorporated PIMD features into the LAMMPS package, allowing for seamless integration with the DeePMD-kit.
- AMBER[141] is integrated with the DeePMD-kit package through the customized source code.[54] The AMBER/DeePMD-kit interface allows for effective QM/MM + DPRc simulations using the DPRc model.[52] The interface extends beyond QM/QM interactions and includes a range correction for QM/MM interactions. The DeePMD-kit package only infers the selected QM region (assigned by an AMBER mask) and its MM buffer within the cutoff radius of the QM region. Like the LAMMPS integration, this interface supports MPI, as discussed in Sec. III B 2, and allows for on-the-fly computation of model deviation during concurrent learning. The AMBER/DeePMD-kit interface also enables alchemical free energy simulations to be performed, leveraging AMBER's GPU-accelerated free energy engine[120] and new features[160–162] for MM transformations and using indirect MM → QM/Δ-MLP methods[163] to correct the end states to the higher level.
- OpenMM,[153] a widely adopted molecular dynamics engine, integrates with the DeePMD-kit through an OpenMM plugin. This plugin enables standard molecular dynamics simulations with DP models and supports hybrid DP/MM-type simulations. In hybrid simulations, the system can be simulated with a fixed DP region or adaptively changing regions during the simulation.[154]
- GROMACS[152] is integrated with the DeePMD-kit through a patch to GROMACS. The patch enables DP/MM simulations by assigning the atom types inferred by DeePMD-kit.
- ABACUS[155] supports the C and C++ interfaces provided by DeePMD-kit. In addition, ABACUS supports various molecular dynamics based on different methods, such as classical molecular dynamics using LJ pair potential and first-principles molecular dynamics based on methods such as Kohn-Sham density functional theory (KSDFT), stochastic density functional theory (SDFT), and orbital-free density functional theory (OFDFT). The possibility of combining the DeePMD-kit with these methods requires further exploration.

These integrations and interfaces with existing packages offer researchers the flexibility to utilize the DeePMD-kit in conjunction with other powerful tools, enhancing the capabilities of molecular dynamics simulations.

2. Customized plugins

DeePMD-kit is built with an object-oriented design, and each component discussed in Sec. II corresponds to a Python class. One of the advantages of this design is the availability of a plugin system for these components. With this plugin system, developers can create and incorporate their customized components, without having to modify the DeePMD-kit package. This approach expedites the realization of their ideas. Moreover, the plugin system facilitates the addition of new components within the DeePMD-kit package itself.

E. DeepModeling community

DeePMD-kit is a free and open-source software licensed under the LGPL-3.0 license, enabling developers to modify and incorporate DeePMD-kit into their own packages. Serving as the core, DeePMD-kit led to the formation of an open-source community named DeepModeling in 2021, which manages open-source packages for scientific computing. Since then, numerous open-source packages for scientific computing have either been created or joined the DeepModeling community, such as DP-GEN,[107] DeePKS-kit,[164] DMFF,[165] ABACUS,[155] DeePH,[166] and DeepFlame,[167] among others, whether directly or indirectly related to DeePMD-kit. The DeepModeling packages that are related to DeePMD-kit are listed as follows.

1. Deep Potential GENerator (DP-GEN)[107] is a package that implements the concurrent learning procedure[103] and is capable of generating uniformly accurate DP models with minimal human intervention and computational cost. DP-GEN2 is the next generation of this package, built on the workflow platform Dflow.
2. Deep Potential Thermodynamic Integration (DP-Ti) is a Python package that enables users to calculate free energy, perform thermodynamic integration, and determine pressure-temperature phase diagrams for materials with DP models.
3. DP-Data is a Python package that helps users convert atomistic data between different formats and calculate atomistic data through electronic calculation and MLP packages. It can be used to generate training data files for DeePMD-kit and visualize structures via 3Dmol.js.[168] The package supports a plugin system and is compatible with ASE,[148] allowing it to support any data format without being limited by the package's code.
4. DP-Dispatcher is a Python package used to generate input scripts for high-performance computing (HPC) schedulers, submit them to HPC systems, and monitor their progress until completion. It was originally developed as part of the DP-GEN package,[107] but has since become an independent package that serves other packages.
5. DArgs is a Python package that manages and filters user input arguments. It provides a Sphinx[142] extension to generate documentation for arguments.

6. DP-GUI is a web-based graphical user interface (GUI) built with the Vue.js framework.[169] It allows users to fill in arguments interactively on a web page and save them to a JSON[145] file. DArgs is used to provide details and documentation of arguments in the GUI.

IV. EXAMPLE APPLICATION: MOLECULAR DYNAMICS

This section introduces a general workflow for performing deep potential molecular dynamics using concurrent learning[170] from scratch, as depicted in Fig. 3. The target simulation can encompass various conditions, such as temperature, pressure, and classical or path-integral dynamics, with or without enhanced sampling methods, in equilibrium or non-equilibrium states, and at different scales and time scales. It is important to note that this section does not serve as a user manual or tutorial or delve into specific systems.

The initial step involves preparing the initial dataset. This dataset is typically generated by sampling from small-scale, short-time MD simulations conducted under the same conditions as target simulations. The simulation level can vary, ranging from *ab initio*[170] to semi-empirical[52] or force fields,[77] depending on the computational cost. Subsequently, these configurations are relabeled using high-accuracy *ab initio* methods.

Once the initial data are ready, the next step involves performing concurrent learning cycles, which are crucial for improving the accuracy of the target simulation. Each cycle comprises three steps: training, exploration, and labeling. In the training step, DeePMD-kit trains multiple models (typically four models) using the existing target data collection with short training steps. These models can be initialized from different random seeds or from the models trained in the previous iteration. In the exploration step, one of the models is employed to perform the target simulation and sample the configurational space. If the target simulation involves a non-equilibrium process, the simulation time can gradually increase with concurrent learning cycles. Configurations (or a subset of atoms within the configurations to reduce computational cost[77]) are randomly selected from configurations that satisfy the following condition:

$$\{R_n | n \in I_{\text{cand}}, I_{\text{cand}} = \{n | \theta_{\text{low}} \le \epsilon_{\text{F,max}} < \theta_{\text{high}}\}\}, \tag{74}$$

where $\epsilon_{\text{F,max}}$ was given in Sec. II C, θ_{low} should be set to a value higher than most of $\epsilon_{\text{F,max}}$ in the existing target data collection, and θ_{high} is typically set to a value ~0.15 eV/Å higher than θ_{low}. These threshold values ensure that only configurations not yet added to the target data collection will be selected. The selected configurations are labeled using consistent *ab initio* methods and added to the target data collection in the labeling step, proceeding to the next iteration.

If the ratio of accurate configurations ($\epsilon_{\text{F,max}} < \theta_{\text{low}}$) in a simulation converges (remains unchanged in subsequent concurrent learning cycles), it can be considered as the target simulation, and the iteration can be stopped. Such a simulation trajectory can be further analyzed.

The above workflow can be executed manually or using the DP-GEN package[107] automatically.

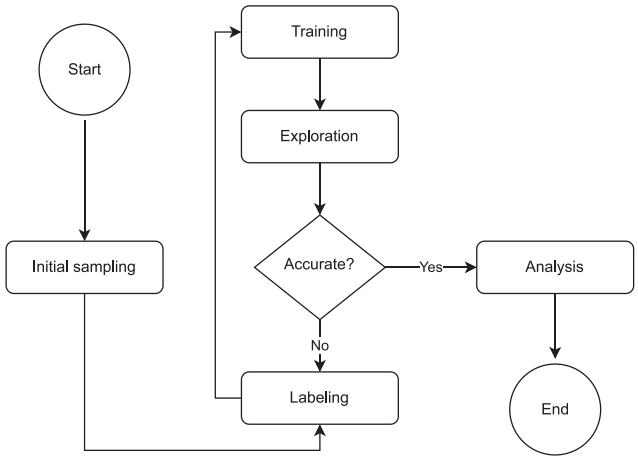

FIG. 3. The general workflow of performing deep potential molecular dynamics in the manner of concurrent learning.

V. BENCHMARKING

We performed benchmarking on various potential energy models with different descriptors on multiple datasets to show the precision and performance of descriptors developed within the DeePMD-kit package. The datasets, the models, the hardware, and the results will be described and discussed in the following Secs. V A–V C.

A. Datasets

The datasets we used included water,[9,61] copper (Cu),[107] high entropy alloys (HEAs),[51,171] OC2M subset in Open Catalyst 2020 (OC20),[115,116] Small-Molecule/Protein Interaction Chemical Energies (SPICEs),[104] and dipeptide subset in SPICE,[104] as shown in Table I and listed as follows:

- The water dataset contains of 140 000 configurations collected from path-integral *ab initio* MD simulations and classical *ab initio* MD simulations for liquid water and ice. Configurations were labeled using the hybrid version of Perdew–Burke–Ernzerhof (PBE0)[172] + Tkatchenko–Scheffler (TS) functional and projector augmented-wave (PAW) method.[173] The energy cutoff was set to 115 Ry (1565 eV).
- The copper dataset consists of 15 366 configurations in Face Centered Cubic (FCC), Hexagonal Close Packed (HCP), and Body Centered Cubic (BCC) crystal. MD simulations sampled the configurations across a temperature range of 50–2579 K and a pressure range of $1–5 \times 10^4$ Bar. The concurrent learning scheme[170] was employed to select the critical configurations that improved the accuracy of an ensemble of models used to estimate the model prediction error. The Perdew–Burke–Ernzerhof (PBE) functional[174] and PAW method were used with an energy cutoff of 650 eV.
- The High Entropy Alloy (HEA) dataset comprises six elements: Ta, Nb, W, Mo, V, and Al.[51,171] These elements

TABLE I. Datasets used to benchmark.

Dataset	No. of frames	Elements	DFT level	References
Water	140 000	H, O	PBE0+TS/PAW (E_{cutoff} = 1565 eV)	9 and 61
Copper	15 366	Cu	PBE/PAW (E_{cutoff} = 650 eV)	107
HEA	8 160	Ta, Nb, W, Mo, V, Al	PBE/PAW(E_{cutoff} = 1200 eV)	51 and 171
OC2M	2 000 000	Ag, Al, As, Au, B, Bi, C, Ca, Cd, Cl, Co, Cr, Cs, Cu, Fe, Ga, Ge, H, Hf, Hg, In, Ir, K, Mg, Mn, Mo, N, Na, Nb, Ni, O, Os, P, Pb, Pd, Pt, Rb, Re, Rh, Ru, S, Sb, Sc, Se, Si, Sn, Sr, Ta, Tc, Te, Ti, Tl, V, W, Y, Zn, Zr	RPBE/PAW (E_{cutoff} = 350 eV)	115 and 116
SPICE	1 132 808	H, Li, C, N, O, F, Na, Mg, P, S, Cl, K, Ca, Br, I	ωB97M-D3BJ/def2-TZVPPD	104
Dipeptides	33 850	H, C, N, O, S	ωB97M-D3BJ/def2-TZVPPD	104

occupy a $2 \times 2 \times 2$ BCC lattice consisting of 16 atoms in a random arrangement. The concentrations of Ta, Nb, W, Mo, and V encompass the entire composition space, while Al is considered an additive, with its maximum quantity being less than six. MD simulations sampled the configurations across a temperature range of 50–388.1 K and a pressure range of $1–5 \times 10^4$ bars. The concurrent learning scheme[170] was employed to select the critical configurations that improved the accuracy of an ensemble of models used to estimate the model prediction error. The dataset comprises 8160 configurations labeled by the density functional theory with PBE approximation[174] of the exchange and correlation. The PAW method was used with an energy cutoff of 1200 eV and a k-space sampling grid size of 0.12 Å^{-1}.

- The OC2M subset[116] in the Open Catalyst 2020 (OC20) dataset takes 2×10^6 configurations from the OC20 dataset[115] and includes 57 elements. OC20 consists of 1 281 040 configurations across a wide swath of materials, surfaces, and adsorbates and is labeled by the revised PBE functional[174] under the periodic boundary condition. The PAW method was employed with an energy cutoff of 350 eV.
- The Small-Molecule/Protein Interaction Chemical Energies (SPICE)[104] dataset is a drug-like dataset that includes various subsets: dipeptides, solvated amino acids, PubChem molecules, DES370K dimers, DES370K monomers, and ion pairs. The dataset is composed of 1 132 808 non-period configurations labeled at the ωB97M-D3BJ/def2-TZVPPD level.[175,176] It consists of 15 elements and contains charged configurations. We adopted the same method as described in Ref. 104 to consider each unique combination of element and formal charge as a different atom type.
- The dipeptide subset in SPICE[104] comprises all possible dipeptides formed by the 20 natural amino acids and their common protonation variants. This subset contains 33 850 configurations with elements, including H, C, N, O, and S, corresponding to the amino acids.

The above datasets are representative as they contain liquids, solids, and gases, configurations in both periodic and non-periodic boundary conditions, configurations spanning a wide range of temperatures and pressures, and ions and drug-like molecules in different protonation states. The study of all these systems is essential in the field of chemical physics.

We split all the datasets into a training set containing 95% of the data and a validation set containing the remaining 5% of the data.

B. Models and hardware

We compared various descriptors, including the local frame (`loc_frame`), two-body embedding full-information DeepPot-SE (`se_e2_a`), a hybrid descriptor with two-body embedding full- and radial-information DeepPot-SE (`se_e2_a+se_e2_r`), a hybrid descriptor with two-body embedding full-information and three-body embedding DeepPot-SE (`se_e2_a+se_e3`), and an attention-based descriptor (`se_atten`). In all models, we set r_s to 0.5 Å, $M_<$ to 16, and L_a to 2, if applicable. We used (25, 50, 100) neurons for two-body embedding networks $\mathcal{N}_{e,2}$, (2, 4, 8) neurons for three-body embedding networks $\mathcal{N}_{e,3}$, and (240, 240, 240, 1) neurons for fitting networks $\mathcal{F}_0$. In the full-information part (`se_e2_a`) of the hybrid descriptor with two-body embedding full-information and radius-information DeepPot-SE (`se_e2_a+se_e2_r`) and the two-body embedding part (`se_e2_a`) of the hybrid descriptor with two-body full-information and three-body DeepPot-SE (`se_e2_a+se_e3`), we set r_c to 4 Å. For the OC2M system, we set r_c to 9 Å, while under other situations, we set r_c to 6 Å.

We trained each model for a fixed number of steps (1 000 000 for water, Cu, and dipeptides, 16 000 000 for HEA, and 10 000 000 for OC2M and SPICE) using neural networks in double floating precision (FP64) and single floating precision (FP32) separately. We used the LAMMPS package[140] to perform MD simulations for water, Cu, and HEA with as many atoms as possible. We compared the performance of compressed models with that of the original model where applicable.[108] The platforms used to benchmark performance included 128-core AMD EPYC 7742, NVIDIA GeForce RTX 3080 Ti (12 GB), NVIDIA Tesla V100 (40 GB), NVIDIA Tesla A100 (80 GB), AMD Instinct MI250, and Xilinx Virtex Ultrascale + VU9P FPGA for NVNMD only.[110] We note that currently, the model compression feature only supports `se_e2_a`, `se_e2_r`, and `se_e3` descriptors, and NVNMD only supports regular `se_e2_a` for systems with no more than four chemical species in FP64

TABLE II. Root mean square errors (RMSEs) in the energy per atom (E, meV/atom) and forces (**F**, meV/Å) for water, Cu, HEA, OC2M, dipeptides, and SPICE validation sets. The boldfaced values denote the best model in an indicator.

		`loc_frame`		`se_e2_a`		`se_e2_a+se_e2_r`		`se_e2_a+se_e3`		`se_atten`	
System	Indicator	FP64	FP32	FP64	FP32	FP64	FP32	FP64	FP32	FP64	FP32
Water	E RMSE	0.7	**0.7**	1.0	1.0	0.9	1.0	1.0	1.0	1.5	1.2
	F RMSE	40.0	**39.2**	49.0	48.4	48.6	50.0	46.5	45.9	44.4	42.3
Cu	E RMSE	12.7	19.2	3.0	2.8	4.8	4.9	**2.5**	2.6	3.2	3.6
	F RMSE	84.7	105	17.7	17.9	21.4	22.0	16.8	**16.6**	16.9	16.9
HEA	E RMSE	···	···	15.4	15.3	13.5	14.5	12.1	17.2	**5.5**	6.4
	F RMSE	···	···	134	137	163	158	136	180	**90.7**	98.3
OC2M	E RMSE	···	···	···	···	···	···	···	···	15.1	**14.3**
	F RMSE	···	···	···	···	···	···	···	···	155	**148**
Dipeptides	E RMSE	···	···	**9.5**	9.6	12.5	11.7	14.9	16.8	12.8	12.7
	F RMSE	···	···	97.9	97.7	98.6	101	160	222	99.7	**96.7**
SPICE	E RMSE	···	···	···	···	···	···	···	···	80.9	**78.3**
	F RMSE	···	···	···	···	···	···	···	···	**233**	234

precision. The model compression feature for `se_atten` is under development.

It is important to note that these models are designed for the purpose of comparing different descriptors and floating-point number precisions supported by the package under the same conditions, with the aim of recommending the best model to use. However, it should be emphasized that the number of training steps is limited, and hyperparameters, such as the number of neurons in neural networks, are not tuned for any specific system. Therefore, it is not advisable to utilize these models for production purposes, and it would be meaningless to compare them with the well-established models reported in other references, Refs. 9, 51, 107, and 104.

Furthermore, it is not recommended to compare these models with models produced by other packages, as it can be challenging to establish a fair comparison. For instance, ensuring that hyperparameters in different models are precisely the same or making all models consume the same computational resources across different packages is not straightforward.

C. Results and discussion

We present the validation errors of different models in Table II, the training and MD performance on various platforms in Tables III and IV, as well as the maximum number of atoms that a platform can simulate in Table V. None of the models outperforms the others in terms of accuracy for all datasets. The non-smooth local frame descriptor achieves the best accuracy for the water system, with an energy RMSE of 0.689 meV/atom and a force RMSE of 39.2 meV/Å. Moreover, this model exhibits the fastest computing performance among all models on CPUs, although it has not yet been implemented on GPUs, as shown in Appendix B. The local frame descriptor, despite having higher accuracy in some cases, has limitations that hinder its widespread applicability. One such limitation is that it is not smooth. Additionally, this descriptor does not perform well for the copper system, which was collected over a wide range of temperatures and pressures.[107] Another limitation is that it requires all systems to have similar chemical species to build the local frame, which makes it challenging to apply in datasets, such as HEA, OC2M, dipeptides, and SPICE.

On the other hand, the DeepPot-SE descriptor offers greater generalization in terms of both accuracy and performance. The compressed models are 1–10× faster than the original for training and inference, and the NVNMD is 50–100× faster than the regular MD, both of which demonstrate impressive computational performance. It is expected that the MD performance of the uncompressed water `se_e2_a` FP64 model (8.25 μs/step/atom on a single V100 card) is close to the MD performance reported in Ref. 41 (8.19 μs/step/atom per V100 card), and the MD performance of the compressed model (1.94 μs/step/atom) is about 3× faster in this case. In addition, the compressed model can simulate 6× atoms in a single card compared to the uncompressed model. The three-body embedding descriptor theoretically contains more information than the two-body embedding descriptor and is expected to be more accurate but slower. While this is true for the water and copper systems, the expected order of accuracy is not clearly observed for the HEA and dipeptides datasets. Further research is required to determine the reason for this discrepancy, but it is likely due to the loss not converging within the same training steps when more chemical species result in more trainable parameters. Furthermore, the performance on these two datasets slows down as there are more neural networks.

TABLE III. Training performance (ms/step) for water, Cu, HEA, OC2M, dipeptides, and SPICE systems. "FP64" means double floating precision, "FP32" means single floating precision, and "FP64c" and "FP32c" mean the compressed training[109] for double and single floating precision, respectively. "EPYC" performed on 128 AMD EPYC 7742 cores, "3080 Ti" performed on an NVIDIA GeForce RTX 3080 Ti card, "V100" performed on an NVIDIA Tesla V100 card, "A100" performed on an NVIDIA Tesla A100 card, and "MI250" performed on an AMD Instinct MI250 Graphics Compute Die (GCD).

		loc_frame		se_e2_a				se_e2_a+se_e2_r				se_e2_a+se_e3				se_atten	
System	Hardware	FP64	FP32	FP64	FP32	FP64c	FP32c	FP64	FP32	FP64c	FP32c	FP64	FP32	FP64c	FP32c	FP64	FP32
Water	EPYC	14.7	9.20	97.3	45.0	28.4	16.2	63.7	32.5	29.9	15.4	141	85.2	34.0	20.6	1210	383
	3080 Ti	7.00	4.80	24.6	10.3	9.70	6.40	26.3	11.6	12.0	8.20	52.8	17.2	16.3	6.80	199	26.9
	V100	7.90	8.50	11.1	8.20	5.90	4.80	13.6	10.9	6.90	6.40	23.5	14.0	8.60	7.30	69.6	31.7
	A100	10.7	10.0	8.20	9.30	4.90	5.70	14.5	10.8	7.80	6.30	24.5	12.0	7.50	7.20	30.8	21.2
	MI250	11.7	10.9	20.3	13.1	7.70	7.00	27.3	19.7	11.5	10.9	278	27.7	12.8	11.2	125	31.7
Cu	EPYC	4.90	3.30	33.7	12.8	8.00	5.40	19.9	10.0	10.5	5.30	45.5	24.2	9.10	6.50	226	89.1
	3080 Ti	3.20	2.20	6.50	5.10	4.60	3.90	8.70	6.30	5.90	3.40	11.8	4.80	7.20	5.70	36.8	8.80
	V100	3.20	3.80	4.20	4.80	3.20	3.70	6.50	5.30	5.50	4.10	7.90	5.60	6.00	5.80	15.6	11.9
	A100	4.00	3.90	3.80	3.70	3.10	3.00	5.40	5.30	4.10	4.10	8.00	5.60	4.80	4.60	11.6	11.2
	MI250	4.80	4.90	6.90	6.40	5.10	5.00	9.10	9.40	7.40	7.00	49.9	10.1	8.00	7.30	23.6	18.6
HEA	EPYC	⋯	⋯	53.4	30.5	19.4	12.2	52.3	29.3	27.7	16.7	83.7	51.1	26.6	15.7	159	60.1
	3080 Ti	⋯	⋯	38.4	25.2	11.2	9.10	71.4	41.8	16.3	12.7	93.6	41.0	19.7	15.0	35.9	9.10
	V100	⋯	⋯	33.2	29.8	11.8	11.1	63.2	47.4	17.5	16.5	65.5	49.6	27.4	18.7	15.6	11.9
	A100	⋯	⋯	30.5	28.6	10.9	10.4	51.6	67.4	16.9	21.2	61.7	52.9	18.6	18.8	11.7	11.5
	MI250	⋯	⋯	48.8	42.7	18.5	18.0	72.3	69.3	28.7	27.3	134	88.4	32.7	32.3	21.6	19.5
OC2M	EPYC	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	2070	625
	3080 Ti	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	352	46.0
	V100	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	120	52.8
	A100	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	51.4	30.9
	MI250	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	171	55.7
Dipeptides	EPYC	⋯	⋯	49.7	30.5	21.2	19.4	52.0	35.3	30.1	21.2	89.5	61.1	35.0	21.2	214	91.5
	3080 Ti	⋯	⋯	54.8	39.5	17.3	11.3	90.0	64.3	19.0	15.3	131	67.7	25.4	19.2	26.1	12.0
	V100	⋯	⋯	54.1	52.6	14.8	14.8	88.0	84.3	20.5	21.7	96.2	103	30.1	30.8	14.3	10.6
	A100	⋯	⋯	50.2	50.8	14.3	14.3	89.0	75.9	20.7	19.9	91.1	82.7	26.6	26.7	13.2	11.1
	MI250	⋯	⋯	66.2	67.8	23.1	22.9	117	112	35.0	32.4	155	129	45.9	44.9	19.6	16.8
SPICE	EPYC	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	244	98.0
	3080 Ti	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	35.4	15.3
	V100	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	17.3	15.9
	A100	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	11.9	12.2
	MI250	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	29.0	24.1

The attention-based models with the type embedding exhibit better accuracy for the HEA system and equivalent accuracy for the dipeptide system. These models also have the advantage of faster training on GPUs, with equivalent accuracy for these two systems, by reducing the number of neural networks. However, this advantage is not observed on CPUs or MD simulations, as attention layers are computationally expensive, which calls for future improvements. Furthermore, when there are many chemical species, the attention-based descriptor requires less CPU or GPU memory than other models since it has fewer neural networks. This feature makes it possible to apply to the OC2M dataset with over 60 species and the SPICE dataset with about 20 species.

It is noteworthy that in nearly all systems, FP32 is 0.5 to 2× faster than FP64 and demonstrates similar validation errors. In this case, since we only apply FP32 into neural networks but keep precision in other components, the model precision is also well-known as "mixed precision." This result is consistent with the fact that mixed precision has been widely adopted in other packages.[38,140] Therefore, FP32 should be widely adopted in most applications. Moreover, FP32 enables high performance on hardware with poor FP64 performance, such as consumer GPUs or CPUs.

TABLE IV. MD performance (μs/step/atom) for water, Cu, and HEA systems. "FP64" means double floating precision, "FP32" means single floating precision, and "FP64c" and "FP32c" mean the compressed model[109] for double and single floating precision, respectively. "EPYC" performed on 128 AMD EPYC 7742 cores, "3080 Ti" performed on an NVIDIA GeForce RTX 3080 Ti card, "V100" performed on an NVIDIA Tesla V100 card, "A100" performed on an NVIDIA Tesla A100 card, "MI250" performed on an AMD Instinct MI250 Graphics Compute Die (GCD), and "VU9P" performed NVNMD[110] on a Xilinx Virtex Ultrascale+ VU9P FPGA board.

		loc_frame		se_e2_a				se_e2_a+se_e2_r				se_e2_a+se_e3				se_atten	
System	Hardware	FP64	FP32	FP64	FP32	FP64c	FP32c	FP64	FP32	FP64c	FP32c	FP64	FP32	FP64c	FP32c	FP64	FP32
Water	EPYC	1.25	0.699	19.3	8.73	3.89	2.61	8.33	3.43	3.78	1.86	37.2	15.1	5.04	3.63	221	83.8
	3080 Ti	12.9	8.63	29.0	4.21	9.71	1.73	20.8	3.43	9.06	1.99	69.5	10.5	18.5	2.89	294	32.3
	V100	16.1	16.8	8.25	4.59	1.94	1.51	6.21	3.53	2.22	1.62	22.2	11.3	3.31	2.41	91.2	37.2
	A100	35.7	33.9	4.37	3.01	1.56	1.42	4.11	2.44	2.07	1.53	12.5	7.17	2.64	2.25	35.6	22.4
	MI250	40.2	39.6	7.74	3.96	1.74	1.41	6.03	3.20	2.00	1.54	30.5	18.8	3.51	2.64	55.0	30.2
	VU9P	...	...	0.306	...	...	...	...	...	...	...	...	...	...	...	...	...
Cu	EPYC	1.14	0.702	22.2	9.38	3.43	2.04	11.9	5.28	3.09	1.56	47.9	19.5	4.20	2.73	200	62.1
	3080 Ti	14.9	8.98	30.5	4.18	8.52	1.51	18.8	3.15	7.98	1.81	74.6	11.2	14.7	2.32	294	33.0
	V100	15.7	15.7	8.73	4.81	1.56	1.27	5.71	3.18	1.84	1.38	24.3	12.2	2.60	1.83	91.1	37.3
	A100	36.9	36.9	4.41	2.65	1.36	1.15	3.35	2.15	1.63	1.42	13.5	7.49	2.15	1.78	36.2	21.0
	MI250	39.0	39.1	8.27	4.13	1.37	1.21	5.62	2.98	1.59	1.35	26.9	12.6	2.56	2.00	55.4	29.5
	VU9P	...	...	0.310	...	...	...	...	...	...	...	...	...	...	...	...	...
HEA	EPYC	...	...	32.8	13.0	7.04	4.58	15.3	7.64	6.83	3.80	81.0	33.4	8.56	5.68	156	45.9
	3080 Ti	...	...	65.3	9.72	10.5	2.51	36.1	6.83	11.9	3.24	171	24.9	29.6	5.37	290	32.8
	V100	...	...	20.1	10.9	2.88	2.39	12.3	6.86	12.3	2.85	55.2	28.4	9.42	5.47	91.2	37.4
	A100	...	...	10.4	6.09	2.13	1.83	7.25	5.48	2.98	2.83	30.1	17.1	4.21	4.22	35.0	20.0
	MI250	...	...	20.1	11.6	4.57	4.22	16.2	12.0	7.01	6.44	76.0	44.9	9.09	7.61	55.7	30.5

TABLE V. The maximum number of atoms (10^3) that a GPU card can simulate for water, Cu, and HEA systems. "FP64" means double floating precision, "FP32" means single floating precision, and "FP64c" and "FP32c" mean the compressed model[109] for double and single floating precision, respectively. "3080 Ti" performed on an NVIDIA GeForce RTX 3080 Ti card (12 GB), "V100" performed on an NVIDIA Tesla V100 card (40 GB), and "A100" performed on an NVIDIA Tesla A100 card (80 GB).[a]

		se_e2_a				se_e2_a+se_e2_r				se_e2_a+se_e3				se_atten	
System	Hardware	FP64	FP32	FP64c	FP32c	FP64	FP32	FP64c	FP32c	FP64	FP32	FP64c	FP32c	FP64	FP32
Water	3080 Ti	27	51	127	141	44	74	94	128	10	21	86	95	3	7
	V100	73	135	415	493	114	196	274	430	28	55	250	265	9	19
	A100	189	332	987	1128	288	488	651	900	76	147	618	736	22	49
Cu	3080 Ti	18	35	214	202	40	77	125	151	7	14	83	144	3	7
	V100	54	106	606	635	122	183	337	461	19	39	271	330	9	19
	A100[b]	141	244	1534	1615	286	453	706	1074	50	99	697	867	22	49
HEA	3080 Ti	11	19	60	69	21	33	47	59	5	9	42	52	3	7
	V100	30	53	175	184	57	87	131	166	13	25	117	142	9	19
	A100	76	132	447	468	140	218	323	408	35	66	292	365	22	48

[a]The results on the MI250 card are not reported since the ROCm Toolkit reported a segmentation fault when the number of atoms increased.
[b]Two MPI ranks were used since integer overflow occurred in TensorFlow when the number of elements in an operator exceeded 2^{31}.

VI. SUMMARY

DeePMD-kit is a powerful and versatile community-developed open-source software package for molecular dynamics (MD) simulations using machine learning potentials (MLPs). Its excellent performance, usability, and extensibility have made it a popular choice for researchers in various fields. DeePMD-kit is licensed under the LGPL-3.0 license, which allows anyone to use, modify, and extend the software freely. Thanks to its well-designed code architecture, DeePMD-kit is highly customizable and can be easily extended in various aspects. The models are organized as Python modules in an object-oriented design and saved into the computing graphs, making it easier to add new models. The com-

puting graph is composed of TensorFlow and customized operators, making it easier to optimize the package for a particular hardware architecture and certain operators. The package also has rich and flexible APIs, making it easier to integrate with other molecular simulation packages. DeePMD-kit is open to contributions from researchers in computational science, and we hope that the community will continue to develop and enhance its features in the future.

ACKNOWLEDGMENTS

The authors thank Yihao Liu, Xinzijian Liu, Haidi Wang, Hailin Yang, and the GitHub user ZhengdQin for their code contribution to DeePMD-kit; Zhi X. Chen, Jincai Yang, and Tong Zhu for suggestions to the manuscript; and Ming Li for designing the highlight image. D.T. is grateful to Stefano Baroni, Riccardo Bertossa, Federico Grasselli, and Paolo Pegolo for enlightening discussions throughout the completion of this work. ChatGPT was used to edit the language with the prompt "polish in English," and its outputs were manually reviewed. The work of J.Z. and D.M.Y. was supported by the National Institutes of Health (Grant No. GM107485 to D.M.Y.) and the National Science Foundation (Grant No. 2209718 to D.M.Y.). J.Z. is grateful for the Van Dyke Award from the Department of Chemistry and Chemical Biology, Rutgers, The State University of New Jersey. The work of Y.C., Yifan Li, and R.C. was supported by the "Chemistry in Solution and at Interfaces" (CSI) Center funded by the United States Department of Energy under Award No. DE-SC0019394. The work of M.R. was supported by the VEGA under Project No. 1/0640/20 and by the Slovak Research and Development Agency under Contract No. APVV-19-0371. The work of Q.Z. was supported by the Science and Technology Innovation Program of Hunan Province under Grant No. 2021RC4026. The work of S.L.B. was supported by the Research Council of Norway through the Centre of Excellence Hylleraas Centre for Quantum Molecular Sciences (Grant No. 262695). The work of C.L. and R.W. was supported by the United States Department of Energy (DOE) under Award No. DE-SC0019759. The work of H.W. was supported by the National Key R&D Program of China under Grant No. 2022YFA1004300 and the National Natural Science Foundation of China under Grant No. 12122103. Computational resources were provided by the Bohrium Cloud Platform at DP technology; the Office of Advanced Research Computing (OARC) at Rutgers, The State University of New Jersey; the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which was supported by National Science Foundation under Grant Nos. 2138259, 2138286, 2138307, 2137603, and 2138296 (supercomputer Expanse at SDSC through allocation Grant No. CHE190067); the Texas Advanced Computing Center (TACC) at the University of Texas at Austin, URL: http://www.tacc.utexas.edu (supercomputer Frontera through allocation Grant No. CHE20002); the AMD Cloud Platform at AMD, Inc; and the Princeton Research Computing resources at Princeton University, which is a consortium of groups led by the Princeton Institute for Computational Science and Engineering (PICSciE) and the Office of Information Technology's Research Computing.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Jinzhe Zeng: Conceptualization (equal); Data curation (equal); Formal analysis (equal); Investigation (equal); Methodology (equal); Project administration (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Duo Zhang**: Data curation (equal); Investigation (equal); Methodology (equal); Software (equal); Writing – original draft (equal); Writing – review & editing (equal). **Denghui Lu**: Methodology (equal); Software (equal); Writing – review & editing (equal). **Pinghui Mo**: Investigation (equal); Methodology (equal); Software (equal); Writing – original draft (equal); Writing – review & editing (equal). **Zeyu Li**: Software (equal); Writing – review & editing (equal). **Yixiao Chen**: Software (equal); Writing – review & editing (equal). **Marián Rynik**: Software (equal); Writing – review & editing (equal). **Li'ang Huang**: Software (equal); Writing – review & editing (equal). **Ziyao Li**: Software (equal); Writing – review & editing (equal). **Shaochen Shi**: Software (equal); Writing – original draft (equal); Writing – review & editing (equal). **Yingze Wang**: Software (equal); Writing – review & editing (equal). **Haotian Ye**: Software (equal); Writing – review & editing (equal). **Ping Tuo**: Software (equal); Writing – review & editing (equal). **Jiabin Yang**: Software (equal); Writing – review & editing (equal). **Ye Ding**: Software (equal); Writing – original draft (equal); Writing – review & editing (equal). **Yifan Li**: Investigation (equal); Software (equal); Writing – review & editing (equal). **Davide Tisi**: Software (equal); Writing – review & editing (equal). **Qiyu Zeng**: Funding acquisition (equal); Software (equal); Writing – review & editing (equal). **Han Bao**: Software (equal); Writing – review & editing (equal). **Yu Xia**: Software (equal); Writing – review & editing (equal). **Jiameng Huang**: Software (equal); Writing – review & editing (equal). **Koki Muraoka**: Software (equal); Writing – review & editing (equal). **Yibo Wang**: Software (equal); Writing – review & editing (equal). **Junhan Chang**: Software (equal); Writing – review & editing (equal). **Fengbo Yuan**: Software (equal); Writing – review & editing (equal). **Sigbjorn Loland Bore**: Funding acquisition (equal); Software (equal); Writing – review & editing (equal). **Chun Cai**: Software (equal); Writing – original draft (supporting); Writing – review & editing (equal). **Yinnian Lin**: Software (equal); Writing – review & editing (equal). **Bo Wang**: Software (equal); Writing – review & editing (equal). **Jiayan Xu**: Software (equal); Writing – review & editing (equal). **Jia-Xin Zhu**: Software (equal); Writing – review & editing (equal). **Chenxing Luo**: Software (equal); Writing – review & editing (equal). **Yuzhi Zhang**: Software (equal); Writing – review & editing (equal). **Rhys E. A. Goodall**: Software (equal); Writing – review & editing (equal). **Wenshuo Liang**: Software (equal); Writing – review & editing (equal). **Anurag Kumar Singh**: Software (equal); Writing – review & editing (equal). **Sikai Yao**: Software (equal); Writing – review & editing (equal). **Jingchao Zhang**: Software (equal); Writing – review & editing (equal). **Renata Wentzcovitch**: Funding acquisition (equal); Supervision (equal); Writing – review & editing (equal). **Jiequn Han**: Methodology (equal);

Software (equal); Writing – review & editing (equal). **Jie Liu**: Conceptualization (equal); Methodology (equal); Resources (equal); Supervision (equal); Writing – review & editing (equal). **Weile Jia**: Conceptualization (equal); Methodology (equal); Supervision (equal); Writing – review & editing (equal). **Darrin M. York**: Conceptualization (equal); Funding acquisition (equal); Methodology (equal); Resources (equal); Supervision (equal); Writing – review & editing (equal). **Weinan E**: Conceptualization (equal); Methodology (equal); Supervision (equal); Writing – review & editing (equal). **Roberto Car**: Conceptualization (equal); Funding acquisition (equal); Methodology (equal); Resources (equal); Supervision (equal); Writing – review & editing (equal). **Linfeng Zhang**: Conceptualization (equal); Methodology (equal); Project administration (equal); Resources (equal); Software (equal); Supervision (equal); Writing – review & editing (equal). **Han Wang**: Conceptualization (equal); Funding acquisition (equal); Methodology (equal); Project administration (equal); Software (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal).

DATA AVAILABILITY

DeePMD-kit is openly hosted at the GitHub repository: https://github.com/deepmodeling/deepmd-kit. The datasets, the models, the simulation systems, and the benchmarking scripts used in this study can be downloaded from the GitHub repository: https://github.com/deepmodeling-activity/deepmd-kit-v2-paper. Other data that support the findings of this study are available from the corresponding author upon reasonable request.

APPENDIX A: FIFTH-ORDER POLYNOMIAL INTERPOLATION

Define a piecewise-defined function $f(x)$, where $f(x)$ is a fifth-order polynomial in the range [0, 1) and is a constant in other intervals,

$$f(x) = \begin{cases} 1, & x < 0, \\ ax^5 + bx^4 + cx^3 + dx^2 + ex + f, & 0 \le x < 1, \\ 0, & x \ge 1. \end{cases} \tag{A1}$$

Let $f(x)$, its first-order derivative $f'(x)$, and its second-order derivative $f''(x)$ be continuous at $x = 0$ and $x = 1$,

$$\begin{cases} \lim_{x\to 0^-} f(x) = \lim_{x\to 0^+} f(x), \\ \lim_{x\to 0^-} f'(x) = \lim_{x\to 0^+} f'(x), \\ \lim_{x\to 0^-} f''(x) = \lim_{x\to 0^+} f''(x), \\ \lim_{x\to 1^-} f(x) = \lim_{x\to 1^+} f(x), \\ \lim_{x\to 1^-} f'(x) = \lim_{x\to 1^+} f'(x), \\ \lim_{x\to 1^-} f''(x) = \lim_{x\to 1^+} f''(x). \end{cases} \tag{A2}$$

Solve Eq. (A2), and the solution is

$$a = -6, \quad b = 15, \quad c = -10, \quad d = 0, \quad e = 0, \quad f = 1. \tag{A3}$$

The final forum of $f(x)$ is

$$f(x) = \begin{cases} 1, & x < 0, \\ -6x^5 + 15x^4 - 10x^3 + 1, & 0 \le x < 1, \\ 0, & x \ge 1. \end{cases} \tag{A4}$$

APPENDIX B: FEATURES WITHOUT GPU SUPPORT

At present, the following features do not have GPU support:

- The local frame descriptor in Eqs. (5)–(10).
- Interpolation with a pairwise potential in Eqs. (52)–(55).
- Calculating the maximum number of neighbors N_c within the cutoff radius from the given data.
- Model deviation in Eqs. (68)–(72).
- All NVNMD-specific features.
- The KSpace solver for DPLR in the LAMMPS plugin.

REFERENCES

[1] J. Behler and M. Parrinello, "Generalized neural-network representation of high-dimensional potential-energy surfaces," Phys. Rev. Lett. **98**, 146401 (2007).
[2] 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).
[3] J. Behler, "Atom-centered symmetry functions for constructing high-dimensional neural network potentials," J. Chem. Phys. **134**, 074106 (2011).
[4] 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).
[5] 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).
[6] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," Nat. Commun. **8**, 13890 (2017).
[7] 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).
[8] X. Chen, M. S. Jørgensen, J. Li, and B. Hammer, "Atomic energies from a convolutional neural network," J. Chem. Theory Comput. **14**, 3933–3942 (2018).
[9] 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).
[10] L. Zhang, J. Han, H. Wang, W. 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 Associates, Inc., 2018), Vol. 31, pp. 4436–4446.
[11] Y. Zhang, C. Hu, and B. Jiang, "Embedded atom neural network potentials: Efficient and accurate machine learning with a physically inspired representation," J. Phys. Chem. Lett. **10**, 4962–4967 (2019).
[12] 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).
[13] 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).
[14] Z. L. Glick, D. P. Metcalf, A. Koutsoukas, S. A. Spronk, D. L. Cheney, and C. D. Sherrill, "AP-Net: An atomic-pairwise neural network for smooth and transferable interaction potentials," J. Chem. Phys. **153**, 044112 (2020).

[15]T. Zubatiuk and O. Isayev, "Development of multimodal machine learning potentials: Toward a physics-aware artificial intelligence," Acc. Chem. Res. **54**, 1575–1585 (2021).
[16]E. R. Khajehpasha, J. A. Finkler, T. D. Kühne, and S. A. Ghasemi, "CENT2: Improved charge equilibration via neural network technique," Phys. Rev. B **105**, 144106 (2022).
[17]X. Pan, J. Yang, R. Van, E. Epifanovsky, J. Ho, J. Huang, J. Pu, Y. Mei, K. Nam, and Y. Shao, "Machine-learning-assisted free energy simulation of solution-phase and enzyme reactions," J. Chem. Theory Comput. **17**, 5745–5758 (2021).
[18]S. Takamoto, C. Shinagawa, D. Motoki, K. Nakago, W. Li, I. Kurata, T. Watanabe, Y. Yayama, H. Iriguchi, Y. Asano, T. Onodera, T. Ishii, T. Kudo, H. Ono, R. Sawada, R. Ishitani, M. Ong, T. Yamaguchi, T. Kataoka, A. Hayashi, N. Charoenphakdee, and T. Ibuka, "Towards universal neural network potential for material discovery applicable to arbitrary combination of 45 elements," Nat. Commun. **13**, 2991 (2022).
[19]A. Musaelian, S. Batzner, A. Johansson, L. Sun, C. J. Owen, M. Kornbluth, and B. Kozinsky, "Learning local equivariant representations for large-scale atomistic dynamics," Nat. Commun. **14**, 579 (2023).
[20]K. T. Schütt, O. T. Unke, and M. Gastegger, "Equivariant message passing for the prediction of tensorial properties and molecular spectra," in *International Conference on Machine Learning* (PMLR, 2021), pp. 9377–9388.
[21]S. Batzner, A. Musaelian, L. Sun, M. Geiger, J. P. Mailoa, M. Kornbluth, N. Molinari, T. E. Smidt, and B. Kozinsky, "E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials," Nat. Commun. **13**, 2453 (2022).
[22]M. Haghighatlari, J. Li, X. Guan, O. Zhang, A. Das, C. J. Stein, F. Heidar-Zadeh, M. Liu, M. Head-Gordon, L. Bertels *et al.*, "NewtonNet: A Newtonian message passing network for deep learning of interatomic potentials and forces," Digital Discovery **1**, 333–343 (2022).
[23]J. Gasteiger, F. Becker, and S. Günnemann, "GemNet: Universal directional graph neural networks for molecules," in *Advances in Neural Information Processing Systems* (Curran Associates, Inc., 2021), Vol. 34, pp. 6790–6802.
[24]C. Chen and S. P. Ong, "A universal graph deep learning interatomic potential for the periodic table," Nat. Comput. Sci. **2**, 718–728 (2022).
[25]R. Drautz, "Atomic cluster expansion for accurate and transferable interatomic potentials," Phys. Rev. B **99**, 014104 (2019).
[26]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).
[27]R. Snyder, B. Kim, X. Pan, Y. Shao, and J. Pu, "Facilitating *ab initio* QM/MM free energy simulations by Gaussian process regression with derivative observations," Phys. Chem. Chem. Phys. **24**, 25134–25143 (2022).
[28]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).
[29]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).
[30]K. T. Schütt, P. Kessel, M. Gastegger, K. A. Nicoli, A. Tkatchenko, and K.-R. Müller, "SchNetPack: A deep learning toolbox for atomistic systems," J. Chem. Theory Comput. **15**, 448–455 (2019).
[31]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).
[32]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).
[33]X. Gao, F. Ramezanghorbani, O. Isayev, J. S. Smith, and A. E. Roitberg, "TorchANI: A free and open source PyTorch-based deep learning implementation of the ANI neural network potentials," J. Chem. Inf. Model. **60**, 3408–3415 (2020).
[34]P. O. Dral, F. Ge, B.-X. Xue, Y.-F. Hou, M. Pinheiro, Jr., J. Huang, and M. Barbatti, "MLatom 2: An integrative platform for atomistic machine learning," Top. Curr. Chem. **379**, 27 (2021).
[35]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).
[36]Y. Zhang, J. Xia, and B. Jiang, "REANN: A PyTorch-based end-to-end multifunctional deep neural network package for molecular, reactive, and periodic systems," J. Chem. Phys. **156**, 114801 (2022).
[37]K. T. Schütt, S. S. P. Hessmann, N. W. A. Gebauer, J. Lederer, and M. Gastegger, "SchNetPack 2.0: A neural network toolbox for atomistic machine learning," J. Chem. Phys. **158**, 144801 (2023).
[38]Z. Fan, Y. Wang, P. Ying, K. Song, J. Wang, Y. Wang, Z. Zeng, K. Xu, E. Lindgren, J. M. Rahm, A. J. Gabourie, J. Liu, H. Dong, J. Wu, Y. Chen, Z. Zhong, J. Sun, P. Erhart, Y. Su, and T. Ala-Nissila, "GPUMD: A package for constructing accurate machine-learned potentials and performing highly efficient atomistic simulations," J. Chem. Phys. **157**, 114801 (2022).
[39]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).
[40]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).
[41]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).
[42]Z. Guo, D. Lu, Y. Yan, S. Hu, R. Liu, G. Tan, N. Sun, W. Jiang, L. Liu, Y. Chen, L. Zhang, M. Chen, H. Wang, and W. Jia, "Extending the limit of molecular dynamics with *ab initio* accuracy to 10 billion atoms," in *PPoPP'22: Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Association for Computing Machinery, New York, 2022), pp. 205–218.
[43]J. Behler, "Perspective: Machine learning potentials for atomistic simulations," J. Chem. Phys. **145**, 170901 (2016).
[44]K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev, and A. Walsh, "Machine learning for molecular and materials science," Nature **559**, 547–555 (2018).
[45]F. Noé, A. Tkatchenko, K.-R. Müller, and C. Clementi, "Machine learning for molecular simulation," Annu. Rev. Phys. Chem. **71**, 361–390 (2020).
[46]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).
[47]M. Pinheiro, Jr., F. Ge, N. Ferré, P. O. Dral, and M. Barbatti, "Choosing the right molecular machine learning potential," Chem. Sci. **12**, 14396–14413 (2021).
[48]S. Manzhos and T. Carrington, Jr., "Neural network potential energy surfaces for small molecules and reactions," Chem. Rev. **121**, 10187–10217 (2021).
[49]J. Zeng, L. Cao, and T. Zhu, "Neural network potentials," in *Quantum Chemistry in the Age of Machine Learning*, edited by P. O. Dral (Elsevier, 2022), Chap. 12, pp. 279–294.
[50]X. Wang, Y. Wang, L. Zhang, F. Dai, and H. Wang, "A tungsten deep neural-network potential for simulating mechanical property degradation under fusion service environment," Nucl. Fusion **62**, 126013 (2022).
[51]D. Zhang, H. Bi, F.-Z. Dai, W. Jiang, L. Zhang, and H. Wang, "DPA-1: Pretraining of attention-based deep potential model for molecular simulation," arXiv.2208.08236 (preprint) (2022).
[52]J. Zeng, T. J. Giese, Ş. Ekesan, and D. M. York, "Development of range-corrected deep learning potentials for fast, accurate quantum mechanical/molecular mechanical simulations of chemical reactions in solution," J. Chem. Theory Comput. **17**, 6993–7009 (2021).
[53]L. Zhang, H. Wang, M. C. Muniz, A. Z. Panagiotopoulos, R. Car, and W. E, "A deep potential model with long-range electrostatic interactions," J. Chem. Phys. **156**, 124107 (2022).
[54]W. Liang, J. Zeng, D. M. York, L. Zhang, and H. Wang, "Learning DeePMD-kit: A guide to building deep potential models," in *A Practical Guide to Recent Advances in Multiscale Modeling and Simulation of Biomolecules*, edited by Y. Wang and R. Zhou (AIP Publishing, 2023), Chap. 6, pp. 1–20.
[55]T. Wen, L. Zhang, H. Wang, W. E, and D. J. Srolovitz, "Deep potentials for materials science," Mater. Futures **1**, 022601 (2022).

[56]S. K. Achar, L. Zhang, and J. K. Johnson, "Efficiently trained deep learning potential for graphane," J. Phys. Chem. C **125**, 14874–14882 (2021).
[57]L. Bonati and M. Parrinello, "Silicon liquid structure and crystal nucleation from *ab initio* deep metadynamics," Phys. Rev. Lett. **121**, 265701 (2018).
[58]J. Wang, H. Shen, R. Yang, K. Xie, C. Zhang, L. Chen, K.-M. Ho, C.-Z. Wang, and S. Wang, "A deep learning interatomic potential developed for atomistic simulation of carbon materials," Carbon **186**, 1–8 (2022).
[59]R. Li, E. Lee, and T. Luo, "A unified deep neural network potential capable of predicting thermal conductivity of silicon in different phases," Mater. Today Phys. **12**, 100181 (2020).
[60]I. A. Balyakin, S. V. Rempel, R. E. Ryltsev, and A. A. Rempel, "Deep machine learning interatomic potential for liquid silica," Phys. Rev. E **102**, 052125 (2020).
[61]H.-Y. Ko, L. Zhang, B. Santra, H. Wang, W. E, R. A. DiStasio, Jr., and R. Car, "Isotope effects in liquid water via deep potential molecular dynamics," Mol. Phys. **117**, 3269–3281 (2019).
[62]J. Xu, C. Zhang, L. Zhang, M. Chen, B. Santra, and X. Wu, "Isotope effects in molecular structures and electronic properties of liquid water via deep potential molecular dynamics based on the SCAN functional," Phys. Rev. B **102**, 214113 (2020).
[63]C. Andreani, G. Romanelli, A. Parmentier, R. Senesi, A. I. Kolesnikov, H.-Y. Ko, M. F. Calegari Andrade, and R. Car, "Hydrogen dynamics in supercritical water probed by neutron scattering and computer simulations," J. Phys. Chem. Lett. **11**, 9461–9467 (2020).
[64]C. Zhang, L. Zhang, J. Xu, F. Tang, B. Santra, and X. Wu, "Isotope effects in x-ray absorption spectra of liquid water," Phys. Rev. B **102**, 115155 (2020).
[65]T. E. Gartner III, L. Zhang, P. M. Piaggi, R. Car, A. Z. Panagiotopoulos, and P. G. Debenedetti, "Signatures of a liquid–liquid transition in an ab initio deep neural network model for water," Proc. Natl. Acad. Sci. U. S. A. **117**, 26040–26046 (2020).
[66]D. Tisi, L. Zhang, R. Bertossa, H. Wang, R. Car, and S. Baroni, "Heat transport in liquid water from first-principles and deep neural network simulations," Phys. Rev. B **104**, 224202 (2021).
[67]C. Malosso, L. Zhang, R. Car, S. Baroni, and D. Tisi, "Viscosity in water from first-principles and deep-neural-network simulations," npj Comput. Mater. **8**, 139 (2022).
[68]Y. Shi, C. C. Doyle, and T. L. Beck, "Condensed phase water molecular multipole moments from deep neural network models trained on *ab initio* simulation data," J. Phys. Chem. Lett. **12**, 10310–10317 (2021).
[69]F. Matusalem, J. Santos Rego, and M. de Koning, "Plastic deformation of superionic water ices," Proc. Natl. Acad. Sci. U. S. A. **119**, e2203397119 (2022).
[70]Y. Zhai, A. Caruso, S. L. Bore, Z. Luo, and F. Paesani, "A 'short blanket' dilemma for a state-of-the-art neural network potential for water: Reproducing experimental properties or the physics of the underlying many-body interactions?," J. Chem. Phys. **158**, 084111 (2023).
[71]S. L. Bore and F. Paesani, "Realistic phase diagram of water from "first principles" data-driven quantum simulations," Nat Commun **14**, 3349 (2023).
[72]J. Zeng, Y. Tao, T. J. Giese, and D. M. York, "QDπ: A quantum deep potential interaction model for drug discovery," J. Chem. Theory Comput. **19**, 1261–1275 (2023).
[73]C. Zhang, S. Yue, A. Z. Panagiotopoulos, M. L. Klein, and X. Wu, "Dissolving salt is not equivalent to applying a pressure on water," Nat. Commun. **13**, 822 (2022).
[74]M. Yang, L. Bonati, D. Polino, and M. Parrinello, "Using metadynamics to build neural network potentials for reactive events: The case of urea decomposition in water," Catal. Today **387**, 143–149 (2022).
[75]T. J. Giese, J. Zeng, Ş. Ekesan, and D. M. York, "Combined QM/MM, machine learning path integral approach to compute free energy profiles and kinetic isotope effects in RNA cleavage reactions," J. Chem. Theory Comput. **18**, 4304–4317 (2022).
[76]J. Liu, R. Liu, Y. Cao, and M. Chen, "Solvation structures of calcium and magnesium ions in water with the presence of hydroxide: A study by deep potential molecular dynamics," Phys. Chem. Chem. Phys. **25**, 983–993 (2023).
[77]J. Zeng, L. Cao, M. Xu, T. Zhu, and J. Z. H. Zhang, "Complex reaction processes in combustion unraveled by neural network-based molecular dynamics simulation," Nat. Commun. **11**, 5713 (2020).
[78]J. Zeng, L. Zhang, H. Wang, and T. Zhu, "Exploring the chemical space of linear alkane pyrolysis via deep potential GENerator," Energy Fuels **35**, 762–769 (2021).
[79]Q. Chu, K. H. Luo, and D. Chen, "Exploring complex reaction networks using neural network-based molecular dynamics simulation," J. Phys. Chem. Lett. **13**, 4052–4057 (2022).
[80]B. Wang, J. Zeng, L. Cao, C.-H. Chin, D. York, T. Zhu, and J. Zhang, "Growth of polycyclic aromatic hydrocarbon and soot inception by in silico simulation," chemrxiv-2022-qp8fc (2022).
[81]Z. Wang, Y. Han, J. Li, and X. He, "Combining the fragmentation approach and neural network potential energy surfaces of fragments for accurate calculation of protein energy," J. Phys. Chem. B **124**, 3027–3035 (2020).
[82]Y. Han, Z. Wang, Z. Wei, J. Liu, and J. Li, "Machine learning builds full-QM precision protein force fields in seconds," Briefings Bioinf. **22**, bbab158 (2021).
[83]M. F. Calegari Andrade, H.-Y. Ko, L. Zhang, R. Car, and A. Selloni, "Free energy of proton transfer at the water-TiO_2 interface from *ab initio* deep potential molecular dynamics," Chem. Sci. **11**, 2335–2341 (2020).
[84]M. Galib and D. T. Limmer, "Reactive uptake of N_2O_5 by atmospheric aerosol is dominated by interfacial processes," Science **371**, 921–925 (2021).
[85]Y.-B. Zhuang, R.-H. Bi, and J. Cheng, "Resolving the odd–even oscillation of water dissociation at rutile TiO_2(110)–water interface by machine learning accelerated molecular dynamics," J. Chem. Phys. **157**, 164701 (2022).
[86]M. de la Puente, R. David, A. Gomez, and D. Laage, "Acids at the edge: Why nitric and formic acid dissociations at air–water interfaces depend on depth and on interface specific area," J. Am. Chem. Soc. **144**, 10524–10529 (2022).
[87]S. P. Niblett, M. Galib, and D. T. Limmer, "Learning intermolecular forces at liquid-vapor interfaces," J. Chem. Phys. **155**, 164101 (2021).
[88]L. Zhang, H. Wang, R. Car, and W. E, "Phase diagram of a deep potential water model," Phys. Rev. Lett. **126**, 236001 (2021).
[89]J. Zeng, Y. Tao, T. J. Giese, and D. M. York, "Modern semiempirical electronic structure methods and machine learning potentials for drug discovery: Conformers, tautomers, and protonation states," J. Chem. Phys. **158**, 124110 (2023).
[90]W.-K. Chen, X.-Y. Liu, W.-H. Fang, P. O. Dral, and G. Cui, "Deep learning for nonadiabatic excited-state dynamics," J. Phys. Chem. Lett. **9**, 6702–6708 (2018).
[91]C. Vega and J. L. F. Abascal, "Simulating water with rigid non-polarizable models: A general perspective," Phys. Chem. Chem. Phys. **13**, 19663–19688 (2011).
[92]E. Sanz, C. Vega, J. L. F. Abascal, and L. G. MacDowell, "Phase diagram of water from computer simulation," Phys. Rev. Lett. **92**, 255701 (2004).
[93]T. Darden, D. York, and L. Pedersen, "Particle mesh Ewald: An N·log(N) method for Ewald sums in large systems," J. Chem. Phys. **98**, 10089–10092 (1993).
[94]T. J. Giese, M. T. Panteva, H. Chen, and D. M. York, "Multipolar Ewald methods, 1: Theory, accuracy, and performance," J. Chem. Theory Comput. **11**, 436–450 (2015).
[95]T. J. Giese, M. T. Panteva, H. Chen, and D. M. York, "Multipolar Ewald methods, 2: Applications using a quantum mechanical force field," J. Chem. Theory Comput. **11**, 451–461 (2015).
[96]K. Nam, J. Gao, and D. M. York, "An efficient linear-scaling Ewald method for long-range electrostatic interactions in combined QM/MM calculations," J. Chem. Theory Comput. **1**, 2–13 (2005).
[97]T. J. Giese and D. M. York, "Ambient-potential composite Ewald method for *ab initio* quantum mechanical/molecular mechanical molecular dynamics simulation," J. Chem. Theory Comput. **12**, 2611–2632 (2016).
[98]T. J. Giese, M. Huang, H. Chen, and D. M. York, "Recent advances toward a general purpose linear-scaling quantum force field," Acc. Chem. Res. **47**, 2812–2820 (2014).
[99]T. J. Giese and D. M. York, "Quantum mechanical force fields for condensed phase molecular simulations," J. Phys.: Condens. Matter **29**, 383002 (2017).
[100]T. J. Giese, J. Zeng, and D. M. York, "Multireference generalization of the weighted thermodynamic perturbation method," J. Phys. Chem. A **126**, 8519–8533 (2022).
[101]T. J. Martínez, "Ab initio reactive computer aided molecular design," Acc. Chem. Res. **50**, 652–656 (2017).

[102]J. Zeng, L. Cao, C.-H. Chin, H. Ren, J. Z. H. Zhang, and T. Zhu, "ReacNetGenerator: An automatic reaction network generator for reactive molecular dynamics simulations," Phys. Chem. Chem. Phys. **22**, 683–691 (2020).
[103]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).
[104]L. Cao, J. Zeng, B. Wang, T. Zhu, and J. Z. Zhang, "*Ab initio* neural network MD simulation of thermal decomposition of high energy material CL-20/TNT," Phys. Chem. Chem. Phys. **24**, 11801–11811 (2022).
[105]L. Zhang, M. Chen, X. Wu, H. Wang, W. E, and R. Car, "Deep neural network for the dielectric response of insulators," Phys. Rev. B **102**, 041121 (2020).
[106]G. M. Sommers, M. F. C. Andrade, L. Zhang, H. Wang, and R. Car, "Raman spectrum and polarizability of liquid water from deep neural networks," Phys. Chem. Chem. Phys. **22**, 10592–10602 (2020).
[107]Y. Zhang, H. Wang, W. Chen, J. Zeng, L. Zhang, W. Han, and W. E, "DP-GEN: A concurrent learning platform for the generation of reliable deep learning based potential energy models," Comput. Phys. Commun. **253**, 107206 (2020).
[108]D. Lu, H. Wang, M. Chen, L. Lin, R. Car, W. E, W. Jia, and L. Zhang, "86 PFLOPS deep potential molecular dynamics simulation of 100 million atoms with *ab initio* accuracy," Comput. Phys. Commun. **259**, 107624 (2021).
[109]D. Lu, W. Jiang, Y. Chen, L. Zhang, W. Jia, H. Wang, and M. Chen, "DP compress: A model compression scheme for generating efficient deep potential models," J. Chem. Theory Comput. **18**, 5559–5567 (2022).
[110]P. Mo, C. Li, D. Zhao, Y. Zhang, M. Shi, J. Li, and J. Liu, "Accurate and efficient molecular dynamics based on machine learning and non von Neumann architecture," npj Comput. Mater. **8**, 107 (2022).
[111]K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision–ECCV 2016* (Springer International Publishing, 2016), pp. 630–645.
[112]V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10)* (Omnipress, Madison, WI, 2010), pp. 807–814.
[113]X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, edited by G. Gordon, D. Dunson, and M. Dudík (Proceedings of Machine Learning Research, Fort Lauderdale, FL, 2011), Vol. 15, pp. 315–323.
[114]D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," arXiv:1606.08415 [cs.LG] (2020).
[115]L. Chanussot, A. Das, S. Goyal, T. Lavril, M. Shuaibi, M. Riviere, K. Tran, J. Heras-Domingo, C. Ho, W. Hu, A. Palizhati, A. Sriram, B. Wood, J. Yoon, D. Parikh, C. L. Zitnick, and Z. Ulissi, "Open catalyst 2020 (OC20) dataset and community challenges," ACS Catal. **11**, 6059–6072 (2021).
[116]J. Gasteiger, M. Shuaibi, A. Sriram, S. Günnemann, Z. Ulissi, C. L. Zitnick, and A. Das, "GemNet-OC: Developing graph neural networks for large and diverse molecular simulation datasets," Transactions on Machine Learning Research (published online, 2022); available at https://dblp.org/db/journals/tmlr/tmlr2022.html [cs.LG] (2022).
[117]A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., 2017), Vol. 30.
[118]M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (Association for Computational Linguistics, Lisbon, Portugal, 2015), pp. 1412–1421
[119]Y. Zhang, C. Gao, Q. Liu, L. Zhang, H. Wang, and M. Chen, "Warm dense matter simulation via electron temperature dependent deep potential molecular dynamics," Phys. Plasmas **27**, 122704 (2020).
[120]T.-S. Lee, D. S. Cerutti, D. Mermelstein, C. Lin, S. LeGrand, T. J. Giese, A. Roitberg, D. A. Case, R. C. Walker, and D. M. York, "GPU-accelerated molecular dynamics and free energy methods in Amber18: Performance enhancements and new features," J. Chem. Inf. Model. **58**, 2043–2050 (2018).
[121]J. Yang, Y. Cong, and H. Li, "A new machine learning approach based on range corrected deep potential model for efficient vibrational frequency computation," arXiv:2303.15969 (2023).
[122]J. F. Ziegler and J. P. Biersack, "The stopping and range of ions in matter," in *Treatise on Heavy-Ion Science: Volume 6: Astrophysics, Chemistry, and Condensed Matter*, edited by D. A. Bromley (Springer, Boston, MA, 1985), pp. 93–129.
[123]H. Wang, X. Guo, L. Zhang, H. Wang, and J. Xue, "Deep learning inter-atomic potential model for accurate irradiation damage simulations," Appl. Phys. Lett. **114**, 244101 (2019).
[124]D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in Proceedings of the 3rd International Conference for Learning Representations (ICLR), 2015.
[125]M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, software available from tensorflow.org.
[126]J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for?," Queue **6**, 40–53 (2008).
[127]AMD, Inc., ROCm—Open Source Platform for HPC and Ultrascale GPU Computing, 2023.
[128]C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," Nature **585**, 357–362 (2020).
[129]A. Collette, *Python and HDF5: Unlocking Scientific Data* (O'Reilly Media, Inc., 2013).
[130]K. Martin and B. Hoffman, Mastering CMake: Version 3.1, Kitware Incorporated, 2015.
[131]J.-C. Fillion-Robin, M. McCormick, O. Padron, M. Smolens, M. Grauer, and M. Sarahan, jcfr/scipy_2018_scikit-build_talk: Scipy 2018 talk—Scikit-build: A build system generator for cpython c/c++/fortran/cython extensions, 2018.
[132]G. Van Rossum and Python Development Team, The Python Library Reference, 12th Media Services, Suwanee, GA, 2018.
[133]Google, Inc., GoogleTest–Google Testing and Mocking Framework, 2023.
[134]L. Dagum and R. Menon, "OpenMP: An industry standard API for shared-memory programming," IEEE Comput. Sci. Eng. **5**, 46–55 (1998).
[135]E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, edited by D. Kranzlmüller, P. Kacsuk, and J. Dongarra (Springer, Berlin, Heidelberg, 2004), pp. 97–104.
[136]W. Gropp, "MPICH2: A new start for MPI implementations," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface* (Springer, Berlin, Heidelberg, 2002), p. 7.
[137]A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," arXiv:1802.05799 (2018).
[138]P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," arXiv:1706.02677 (2017).
[139]L. Dalcin and Y.-L. L. Fang, "mpi4py: Status update after 12 years of development," Comput. Sci. Eng. **23**, 47–54 (2021).
[140]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).
[141]D. A. Case, K. Belfon, I. Y. Ben-Shalom, S. R. Brozell, D. S. Cerutti, T. E. Cheatham III, V. W. D. Cruzeiro, T. A. Darden, R. E. Duke, G. Giambasu, M. K. Gilson, H. Gohlke, A. W. Goetz, R. Harris, S. Izadi, S. A. Izmailov, K. Kasavajhala,

K. Kovalenko, R. Krasny, T. Kurtzman, T. Lee, S. Le-Grand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, V. Man, K. Merz, Y. Miao, O. Mikhailovskii, G. Monard, , H. Nguyen, A. Onufriev, F. Pan, S. Pantano, R. Qi, D. R. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, J. Shen, C. L. Simmerling, N. Skrynnikov, J. Smith, J. Swails, R. C. Walker, J. Wang, R. M. Wilson, R. M. Wolf, X. Wu, Y. Xiong, Y. Xue, D. M. York, and P. A. Kollman, AMBER 20, University of California, San Francisco, CA, 2020.

[142]T. Komiya, G. Brandl, T. Shimizukawa, J. L. Andersen, A. Turner, S. Finucane, R. Lehmann, T. Kampik, J. Magin, J. Dufresne, J. Waltman, J. L. C. Rodríguez, A. Ronacher, M. Geier, D. Shachnev, R. Ruana, P. Virtanen, F. Freitag, L. Maddox, M. Liška, H. Xu, E. Wieser, J. Maitin-Shepard, N. Kaneko, and cocoatomo, sphinx-doc/sphinx: v7.0.0, 2023.

[143]D. van Heesch, Doxygen: Source Code Documentation Generator Tool, 2022.

[144]Conda-Forge Community, The Conda-Forge Project: Community-Based Software Distribution Built on the Conda Package Format and Ecosystem, 2015.

[145]F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of JSON schema," in *Proceedings of the 25th International Conference on World Wide Web* (International World Wide Web Conferences Steering Committee, 2016), pp. 263–273.

[146]O. Ben-Kiki, C. Evans, and I. döt Net, "YAML Ain't Markup Language (YAML™) version 1.2, 2009.

[147]Q. Koziol and D. Robinson, HDF5, 2018.

[148]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).

[149]W. Laosi, H. Gao, Y. Han, C. Ding, P. Shuning, Y. Wang, J. Qiuhan, H.-T. Wang, D. Xing, and J. Sun, "MAGUS: Machine learning and graph theory assisted universal structure searcher," Natl. Sci. Rev. **10**, nwad128 (2023).

[150]S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," J. Comput. Phys. **117**, 1–19 (1995).

[151]V. Kapil, M. Rossi, O. Marsalek, R. Petraglia, Y. Litman, T. Spura, B. Cheng, A. Cuzzocrea, R. H. Meißner, D. M. Wilkins, B. A. Helfrecht, P. Juda, S. P. Bienvenue, W. Fang, J. Kessler, I. Poltavsky, S. Vandenbrande, J. Wieme, C. Corminboeuf, T. D. Kühne, D. E. Manolopoulos, T. E. Markland, J. O. Richardson, A. Tkatchenko, G. A. Tribello, V. V. Speybroeck, and M. Ceriotti, "i-PI 2.0: A universal force engine for advanced molecular simulations," Comput. Phys. Commun. **236**, 214–223 (2019).

[152]M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," SoftwareX **1–2**, 19–25 (2015).

[153]P. Eastman, J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern *et al.*, "OpenMM 7: Rapid development of high performance algorithms for molecular dynamics," PLoS Comput. Biol. **13**, e1005659 (2017).

[154]Y. Ding and J. Huang (2023). "Implementation and validation of an openmm plugin for the deep potential representation of potential energy," https://github.com/JingHuangLab/openmm_deepmd_plugin.

[155]P. Li, X. Liu, M. Chen, P. Lin, X. Ren, L. Lin, C. Yang, and L. He, "Large-scale *ab initio* simulations based on systematically improvable atomic basis," Comput. Mater. Sci. **112**, 503–517 (2016).

[156]P. M. Piaggi, J. Weis, A. Z. Panagiotopoulos, P. G. Debenedetti, and R. Car, "Homogeneous ice nucleation in an ab initio machine-learning model of water," Proc. Natl. Acad. Sci. U. S. A. **119**, e2207294119 (2022).

[157]S. K. Achar, J. J. Wardzala, L. Bernasconi, L. Zhang, and J. K. Johnson, "Combined deep learning and classical potential approach for modeling diffusion in UiO-66," J. Chem. Theory Comput. **18**, 3593–3606 (2022).

[158]P. P. Ewald, "Die berechnung optischer und elektrostatischer gitterpotentiale," Ann. Phys. **369**, 253–287 (1921).

[159]T. A. Barnes, E. Marin-Rimoldi, S. Ellis, and T. D. Crawford, "The MolSSI driver interface project: A framework for standardized, on-the-fly interoperability between computational molecular sciences codes," Comput. Phys. Commun. **261**, 107688 (2021).

[160]H.-C. Tsai, T.-S. Lee, A. Ganguly, T. J. Giese, M. C. Ebert, P. Labute, K. M. Merz, Jr., and D. M. York, "AMBER free energy tools: A new framework for the design of optimized alchemical transformation pathways," J. Chem. Theory Comput. **19**, 640–658 (2023).

[161]T.-S. Lee, H.-C. Tsai, A. Ganguly, and D. M. York, "ACES: Optimized alchemically enhanced sampling," J. Chem. Theory Comput. **19**, 472–487 (2023).

[162]A. Ganguly, H.-C. Tsai, M. Fernández-Pendás, T.-S. Lee, T. J. Giese, and D. M. York, "AMBER drug discovery boost tools: Automated workflow for production free-energy simulation setup and analysis (ProFESSA)," J. Chem. Inf. Model. **62**, 6069–6083 (2022).

[163]T. J. Giese and D. M. York, "Development of a robust indirect approach for MM → QM free energy calculations that combines force-matched reference potential and Bennett's acceptance ratio methods," J. Chem. Theory Comput. **15**, 5543–5562 (2019).

[164]Y. Chen, L. Zhang, H. Wang, and W. E, "DeePKS-kit: A package for developing machine learning-based chemically accurate energy and density functional models," Comput. Phys. Commun. **282**, 108520 (2023).

[165]X. Wang, J. Li, L. Yang, F. Chen, Y. Wang, J. Chang, J. Chen, L. Zhang, and K. Yu, "DMFF: An open-source automatic differentiable platform for molecular force field development and molecular dynamics simulation," Physical Chemistry (published online, 2022).

[166]H. Li, Z. Wang, N. Zou, M. Ye, R. Xu, X. Gong, W. Duan, and Y. Xu, "Deep-learning density functional theory Hamiltonian for efficient ab initio electronic-structure calculation," Nat. Comput. Sci. **2**, 367–377 (2022).

[167]R. Mao, M. Lin, Y. Zhang, T. Zhang, Z.-Q. J. Xu, and Z. X. Chen, "DeepFlame: A deep learning empowered open-source platform for reacting flow simulations," Computer Physics Communications (to be published).

[168]N. Rego and D. Koes, "3Dmol.js: Molecular visualization with WebGL," Bioinformatics **31**, 1322–1324 (2015).

[169]E. You, Vue.js—The Progressive JavaScript Framework, 2023.

[170]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).

[171]W. Jiang, D. Zhang, S. Yao, L. Zhang, H. Wang, and F. Dai, "Hybrid Monte Carlo-molecular dynamics simulation of order-disorder transition in refractory high entropy alloys using deep potential model reliable in the full concentration space" (unpublished) (2023).

[172]C. Adamo and V. Barone, "Toward reliable density functional methods without adjustable parameters: The PBE0 model," J. Chem. Phys. **110**, 6158–6170 (1999).

[173]P. E. Blöchl, "Projector augmented-wave method," Phys. Rev. B **50**, 17953 (1994).

[174]J. P. Perdew, K. Burke, and M. Ernzerhof, "Generalized gradient approximation made simple," Phys. Rev. Lett. **77**, 3865 (1996).

[175]N. Mardirossian and M. Head-Gordon, "ωB97M-V: A combinatorially optimized, range-separated hybrid, meta-GGA density functional with VV10 nonlocal correlation," J. Chem. Phys. **144**, 214110 (2016).

[176]A. Najibi and L. Goerigk, "DFT-D4 counterparts of leading meta-generalized-gradient approximation and hybrid density functionals for energetics and geometries," J. Comput. Chem. **41**, 2562–2572 (2020).