Computer Physics Communications 253 (2020) 107206

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

DP-GEN: A concurrent learning platform for the generation of reliable deep learning based potential energy models☆,☆☆

Yuzhi Zhang [a,b], Haidi Wang [c], Weijie Chen [d], Jinzhe Zeng [e], Linfeng Zhang [f,*], Han Wang [g,*], Weinan E [a,f,*]

[a] Beijing Institute of Big Data Research, Beijing 100871, People's Republic of China
[b] Yuanpei College of Peking University, Beijing 100871, People's Republic of China
[c] School of Electronic Science and Applied Physics, Hefei University of Technology, Hefei 230601, People's Republic of China
[d] Academy for Advanced Interdisciplinary Studies, Peking University, Beijing 100871, People's Republic of China
[e] School of Chemistry and Molecular Engineering, East China Normal University, Shanghai 200062, People's Republic of China
[f] Program in Applied and Computational Mathematics, Princeton University, Princeton, NJ, USA
[g] Laboratory of Computational Physics, Institute of Applied Physics and Computational Mathematics, Huayuan Road 6, Beijing 100088, People's Republic of China

ARTICLE INFO

Article history:
Received 4 November 2019
Received in revised form 25 January 2020
Accepted 31 January 2020
Available online 11 February 2020

Keywords:
Many-body potential energy
Deep learning
Concurrent learning

ABSTRACT

In recent years, promising deep learning based interatomic potential energy surface (PES) models have been proposed that can potentially allow us to perform molecular dynamics simulations for large scale systems with quantum accuracy. However, making these models truly reliable and practically useful is still a very non-trivial task. A key component in this task is the generation of datasets used in model training. In this paper, we introduce the Deep Potential GENerator (DP-GEN), an open-source software platform that implements the recently proposed "on-the-fly" learning procedure (Zhang et al. 2019) and is capable of generating uniformly accurate deep learning based PES models in a way that minimizes human intervention and the computational cost for data generation and model training. DP-GEN automatically and iteratively performs three steps: exploration, labeling, and training. It supports various popular packages for these three steps: LAMMPS for exploration, Quantum Espresso, VASP, CP2K, etc. for labeling, and DeePMD-kit for training. It also allows automatic job submission and result collection on different types of machines, such as high performance clusters and cloud machines, and is adaptive to different job management tools, including Slurm, PBS, and LSF. As a concrete example, we illustrate the details of the process for generating a general-purpose PES model for Cu using DP-GEN.

Program summary

Program Title: DP-GEN
Program Files doi: http://dx.doi.org/10.17632/sxybkgc5xc.1
Licensing provisions: LGPL
Programming language: Python
Nature of problem: Generating reliable deep learning based potential energy models with minimal human intervention and computational cost.
Solution method: The concurrent learning scheme is implemented. Supports for sampling configuration space with LAMMPS, generating *ab initio* data with Quantum Espresso, VASP, CP2K and training potential models with DeePMD-kit are provided. Supports for different machines including workstations, high performance clusters and cloud machines are provided. Supports for job management tools including Slurm, PBS, LSF are provided.

1. Introduction

In recent years, machine learning (ML) has emerged as a promising tool for the field of molecular modeling. In particular, ML-based models have been proposed to address a long-standing issue, the accuracy-*vs*-efficiency dilemma when one evaluates the potential energy surface (PES), a function of atomic positions and their chemical species, and its negative gradients with respect

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

☆☆ The review of this paper was arranged by Prof. Stephan Fritzsche.

* Corresponding authors.
E-mail addresses: linfengz@princeton.edu (L. Zhang), wang_han@iapcm.ac.cn (H. Wang), weinan@math.princeton.edu (W. E).

https://doi.org/10.1016/j.cpc.2020.107206
0010-4655/

to the atomic positions, namely the interatomic forces. From a first-principles point of view, PES is derived from the many-particle Schrödinger equation under the Born–Oppenheimer approximation, and the interatomic forces are given naturally by the Hellman–Feynman theorem. To this end, the *ab initio* molecular dynamics (AIMD) scheme, wherein accurate PES and interatomic forces are obtained within the density functional theory (DFT) approximation, has been most widely adopted [1–3]. Unfortunately, the cost of AIMD restricts its typical applications to system sizes of hundreds of atoms and the time scale of $\sim$100 ps. In the opposite direction, efficient empirical PES models, or force fields (FF), allow us to perform much larger and longer simulations; but their accuracy and transferability is often an issue. ML has the potential to change this situation: a good ML model trained on *ab initio* data should have an efficiency comparable with that of FF models in the sense that the costs of ML models and FF models both scale linearly with system size, while maintaining *ab initio* accuracy.

Developing ML-based PES models involves two components, data generation and model construction. To date, most discussions have focused on the second component. Two important issues are: A good functional form (e.g. kernel based models or neural networks) and respecting physical constraints of the PES, such as the extensiveness and symmetry properties. In this regard, two representative classes of models have emerged: the kernel-based models like the Gaussian Approximation Potential [4] and the neural network (DNN) based models like the Behler–Parrinello model [5] and the Deep Potential model [6,7]. In particular, the smooth version of the Deep Potential model is an end-to-end model that satisfies the requirements mentioned above [8].

There have also been some efforts on open-source software along this line [9–12]. Of particular relevance to this work is the DeePMD-kit package [10], which has been developed to minimize the effort required to build DNN-based PES models and to perform DNN-based MD simulation. DeePMD-kit is interfaced with TensorFlow [13], one of the most popular deep learning frameworks, making the training process highly automatic and efficient. DeePMD-kit is also interfaced with popular MD packages, such as the LAMMPS package [14] for classical MD and the i-PI package [15] for path integral MD. Thus, once one has a good sets of data, there are now effective tools for training Deep Potentials that can be readily used to perform efficient molecular dynamics simulation for all different kinds of purposes.

In comparison, much less effort has gone into the first component mentioned above: data generation. In spite of the tremendous interest and activity, very few have taken the effort to make sure that the dataset used to train the ML-based PES is truly representative enough. Indeed data generation is often quite *ad hoc*. Some notable exceptions are found in [16–18]. In Ref. [17], an active learning procedure was proposed based on an existing unlabeled dataset. Some data points in that set are selected to be labeled, and the result is then used to train the ML model. The procedure ensures that the selected dataset is at least representative of the original unlabeled dataset. In Refs. [16,18], one begins with no data, labeled or unlabeled, and explores the configuration spaces following some systematic procedure. For each of the configurations encountered, a decision is made as to whether that configuration should be labeled. The exploration procedure is designed to ensure that the unlabeled dataset, i.e. all the configurations encountered in the exploration, is representative of all the situations that the ML-based model is intended for. Even though these procedures were also described as "active learning", there is a difference since in these procedures, one does not have the unlabeled data to begin with and choosing the unlabeled data judiciously is also an important part of the whole procedure.

To highlight this difference, we will call the procedures in Refs. [16,18] "concurrent learning". By "concurrent learning", we mean that one does not have any data to begin with, labeled or unlabeled, and the data is generated on the fly as the training proceeds. The generation of the data and the learning is an interactive process to ensure that one obtains an "optimal" dataset which is on one hand representative enough and on the other hand as small as possible. This is in contrast to "sequential learning" in which the data is generated beforehand and the training of the model is performed afterwards. It also differs from active learning in the sense that active learning starts with unlabeled data. In a way the purpose of active learning is to find the smallest dataset that needs to be labeled in an existing unlabeled dataset.

The actual concurrent learning procedure goes as follows. One uses different sampling techniques (such as direct MD at different thermodynamic conditions, enhanced sampling, Monte Carlo) based on the current approximation of the PES to explore the configuration space. An efficient error indicator (this is the error in the PES) is then used to monitor the snapshots generated during the sampling process. Those that have significant errors will then be selected and sent to a labeling procedure, in which accurate *ab initio* energies and forces are calculated and added to the training dataset. A new approximation of the PES is obtained by training with the accumulated training dataset. These steps are repeated until convergence is achieved, i.e., the configuration space has been explored sufficiently, and a representative set of data points has been accurately labeled. At the end of this procedure, a uniformly accurate PES model is generated. We refer to Ref. [18] for more details.

In order to carry out such a procedure efficiently, one also needs reasonable computational resources. Since many tasks can be done in parallel, one needs to implement automatic and efficient parallel processing algorithms. Taking the exploration stage for example, it may happen that dozens to hundreds of MD simulations are executed simultaneously with different initial configurations under different thermodynamic conditions. If these tasks are executed manually, it will require a great deal of human labor, not to mention the compromise in efficiency. It would be even worse if one wants to utilize different computational resources in different concurrent learning steps, e.g., a high performance cluster (HPC) with most advanced GPU nodes for training and an HPC with a vast number of CPU nodes for labeling. Selecting a machine with the most available computational power among a group of candidate machines is also an issue. For all these reasons, we feel that it would be useful for the molecular and materials simulation community to have an open-source implementation of the concurrent learning procedure which, among other things, can automatically schedule the iterative process, dispatch different computational tasks to different computational resources, and collect and analyze the results.

In this paper, we introduce DP-GEN, an open-source concurrent learning platform and software package for the generation of reliable deep learning based PES models, in a way that minimizes the computational cost and human intervention. We describe the implementation of DP-GEN, which is based on the procedure proposed in Ref. [18]. We will focus on two modules, the scheduler and the task dispatcher. A modularized coding structure for the scheduler is designed, making it possible to incorporate different methods or software packages for the three different components in the concurrent learning procedure: exploration, labeling, and training. The dispatcher module is prepared for handling a huge number of tasks in a high-throughput fashion, and it is made compatible with different kinds of machines and popular job scheduling systems.

This paper is organized as follows. In Section 2 we present the basic methodology that the DP-GEN workflow follows. In

Section 3 we introduce the details of the software, including how the concurrent learning process is scheduled and how different tasks are dispatched. In Section 4, we give a concrete example, in which a general purpose Deep Potential model for Cu is generated using DP-GEN. Conclusions and outlooks are given in the last Section.

2. Methodology

The DP-GEN workflow contains a series of successive iterations. Each iteration is composed of three steps: exploration, labeling, and training. We denote by $E_\omega(\mathcal{R})$, abbreviated E_ω, the PES represented by the DP model, where $\mathcal{R}$ denotes atomic positions and ω denotes the parameters. An important point throughout the DP-GEN procedure is that we have an ensemble of models $\{E_{\omega_1}, E_{\omega_2}, \dots, E_{\omega_\alpha}, \dots\}$ trained from the same set of data but with difference in the initialization of model parameters ω_α. ω_α evolves during the training process, which is designed to minimize the loss function, a measure of the error between DP and DFT results for the energies, the forces, and/or the virial tensors. Since the loss function is a non-convex function of ω, such a difference of initialization leads to different minimizers after training, and therefore different PES models. Around configurations where there is enough training data, the different PES models should all be quite accurate and therefore produce predictions that are close to each other. Otherwise one would expect that the predictions from the different models will scatter with a considerable variance. Therefore, the variance of the ensemble of predictions from the ensemble of models for a particular configuration can be used as an error indicator and criterion for whether the configuration should be selected for labeling.

Exploration: This has two components: an efficient sampler and an error indicator. The goal of the sampler is to efficiently explore the configuration space. Assume that we have an initial configuration denoted by $\mathcal{R}_0$. The sampler, in general, is a propagator that evolves an initial configuration of a system to a series of new configurations,

$$\mathcal{R}_t = \varphi_t(\mathcal{R}_0, E_\omega), \tag{1}$$

where φ_t can be either deterministic or stochastic, with t labeling a continuous, or discrete, series of operations, and E_ω indicates that the DP model is parameterized by ω. Available implementations of the sampler include direct MD, MD with enhanced sampling techniques, the Markov chain Monte Carlo (MCMC) approach, and the genetic algorithm (GA), etc. Both the sampler and the initial configurations should be chosen to ensure that all configurations of practical interest are approximately visited with sufficiently high frequency. It is worth noting that since the sampler (1) uses the DP model – rather than an *ab initio* routine – to evaluate potential energies and atomic forces, the exploration is significantly more efficient than AIMD simulations.

Next, given a configuration $\mathcal{R}_t$, we define the error indicator ϵ_t as the maximal standard deviation of the atomic force predicted by the model ensemble, i.e.,

$$\epsilon_t = \max_i \sqrt{\langle \|F_{w,i}(\mathcal{R}_t) - \langle F_{w,i}(\mathcal{R}_t)\rangle\|^2\rangle} \tag{2}$$

where $F_{w,i}(\mathcal{R}_t) = -\nabla_i E_w(\mathcal{R}_t)$ denotes the force on the atom with index i predicted by the model E_w, and ∇_i denotes the derivative with respect to the coordinate of the ith atom. Both of the notations $\langle\dots\rangle$ in Eq. (2) denote the expectation with respect to the ensemble of models, and are estimated by the average of model predictions. For example, $\langle F_{w,i}(\mathcal{R}_t)\rangle$ is estimated by

$$\langle F_{w,i}(\mathcal{R}_t)\rangle = \frac{1}{N_m}\sum_{\alpha=1}^{N_m} F_{w_\alpha,i}(\mathcal{R}_t) \tag{3}$$

Due to the way the error indicator is defined, we also call it the model deviation. The reason for using the predicted forces, rather than energies, to evaluate the model deviation is that the force is an atomic property and is sensitive to the local accuracy, especially when a failure happens. Energy is a global quantity and does not seem to provide sufficient resolution. However, the model deviation is still a lower bound of the real prediction error of the forces. The mathematical derivation and numerical results are given in Appendix B.

As shown in Fig. 1, given an upper and lower bound of the trust levels, σ_{hi} and σ_{lo}, those structures whose model deviations ϵ fall between the bounds will be considered candidates for labeling. Thus, the candidate set of configurations is defined by

$$\{\mathcal{R}_{n\Delta t} | n \in I_{\text{cand}}, \quad I_{\text{cand}} = \{\, n \,|\sigma_{lo} \le \epsilon_{n\Delta t} < \sigma_{hi}\}\}. \tag{4}$$

DP-GEN will then randomly select user-defined number of configurations from the candidate set, and pass them into the next step.

To better illustrate the idea, we take the copper system as an example, for which the user targets at a uniform accuracy over the thermodynamic range of temperature from 0 K to $2T_m$ and pressure from 1 Bar to 50,000 Bar, where $T_m = 1357.77$ K denotes the melting point at the ambient condition. The exploration strategy can be designed as follows. We divide the temperature range $[50\text{ K}, 2T_m)$ equally into four intervals, i.e. $[50\text{ K}, 0.5T_m)$, $[0.5T_m, T_m)$, $[T_m, 1.5T_m)$, $[1.5T_m, 2T_m)$, and explore them successively. In each temperature interval we run 8 iterations with increasing number of MD simulations and increasing length of trajectories. For example in the first iteration, 600 MD simulations of 2 ps are performed, while in the last iteration the number of MD simulation increases to 2400 and the length trajectory increases to 6 ps. In each iteration, 5 temperatures conditions that evenly divide the temperature interval and 8 pressure conditions, i.e. 1, 10 100, 1000, 5000, 10,000, 20,000 and 50,000 Bar, are explored simultaneously by Isothermal–isobaric (NPT) DP-based MD simulations. The initial configurations of these MD simulations are prepared by randomly perturbing fully relaxed standard crystal structures, including face-centered cubic (fcc), hexagonal-closed-packed (hcp), and body-centered cubic (bcc) structures. The configurations along MD trajectories are recorded at a time interval of $\Delta t = 0.02$ ps, and those with model deviation $0.05\text{ eV/Å} \le \epsilon_{n\Delta t} < 0.20\text{ eV/Å}$ are selected and passed to the labeling stage. General instructions on the settings of the bounds can be referred to the end of this section.

Labeling: This step calculates the reference *ab initio* energies $\tilde{E}$ and forces $\tilde{F}$ for the selected configurations $\mathcal{R}_{n\Delta t}$ from the *exploration* step. The process can be done by first-principles-based schemes, including quantum chemistry, quantum Monte Carlo, and DFT, etc. The results are called labels. These labels are then added into the training dataset, to be used later for retraining.

Training: We adopt an advanced version of the Deep Potential (DP) method proposed in Ref. [8]. The DP model considers E_ω as a sum of contributions from atomic local environments $\mathcal{R}^i$, i.e. , $E_\omega = \sum_i E^i_\omega(\mathcal{R}^i)$. $\mathcal{R}^i$ contains the coordinates of i's neighboring atoms within a r_c cutoff radius, and it is mapped, through an embedding network, onto a so-called feature matrix $\mathcal{D}^i_\omega$. Such a construction guarantees the invariance of the PES under translation, rotation, and permutation among identical particles. $\mathcal{D}^i_\omega$ is then mapped, through a fitting network, to E^i_ω. Since ω is composed of the parameters in the embedding network and the fitting network, we also call it the network parameters.

We notice that while the training and labeling steps are quite general, in the DP-GEN workflow, two components in the exploration step require human intervention. First, since in the very beginning, no DP models are available, one needs to explore a set

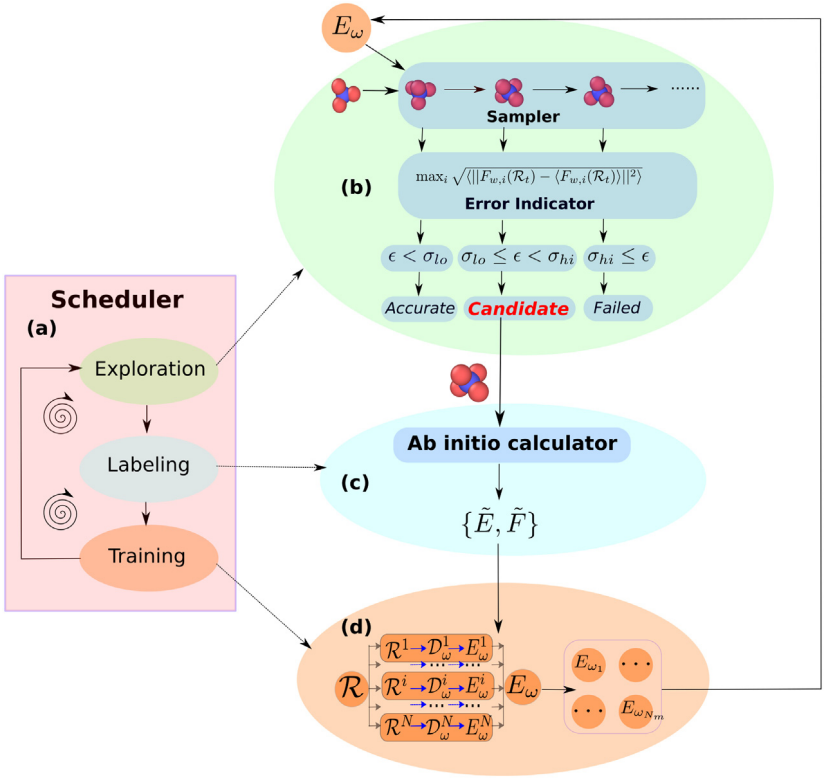

Fig. 1. Schematic illustration of the DP-GEN scheme. (a) The scheduler iteratively and automatically promotes exploration, labeling and training steps. (b) An exploration strategy based on DPMD is taken as an example. Given fixed structures as starting points, an ensemble of DP models E_ω is used to drive MD simulations and sample a series of configurations. For each configuration, the error indicator ϵ, defined by the max force deviation of atomic forces predicted by the DP model ensemble, is calculated. Only those satisfying the criterion $\sigma_{lo} \le \epsilon < \sigma_{hi}$ are selected as candidates for labeling. (c) In the labeling step, the *ab initio* calculator computes first-principles energies $\tilde{E}$ and forces $\tilde{F}$ for the candidates. (d) Based on the expanded training data set, a new ensemble of DP models is obtained and passed to the next iteration.

of initial configurations and do the labeling to kick-off the DP-GEN process. The preparation of the initial configurations could be case-specific. Second, the exploration strategies adopted in each iteration may depend on the purpose of different studies. In practice, we find that the resulting DP models are not very sensitive to the initial configurations as long as they represent reasonable starting points and are close to initial conditions of later explorations. Moreover, in the software, we provide flexible interfaces for users to use different exploration strategies.

The settings of bounds of trust levels σ_{lo} and σ_{hi} deserve careful consideration. Recall that in the exploration step, all configurations $\mathcal{R}_t$ with model deviation being ϵ_t will be categorized into the following three types.

- $\epsilon_t < \sigma_{lo}$. This means that the majority of the explored configurations can be predicted with a high accuracy by the DP models. If the configurations are randomly selected, then most of them are already accurate under the current model. Labeling these configurations does not help improving the quality of the model, but lead to a waste of the computational resources. The lower bound of trust level σ_{lo} thus serves to exclude those configurations which have been predicted accurately by the current models from the configurations to be calculated with first-principles. While in principle σ_{lo} should be set to the desired accuracy of the model, in practice the accuracy of the model is lower bounded by the numerical error in first-principles calculation (finite plane wave cut-off, finite k-space integration grid, and so on) and the fitting ability of the deep potential. Thus, σ_{lo} should not be set arbitrarily small, but a value slightly higher than the training accuracy achievable by DP.
- $\sigma_{hi} \le \epsilon_t$. It illustrates that the exploration is driven by a relatively poor model. This situation often happens in the first several iterations, where DPMD simulations sometimes sample unphysical configurations with, e.g., overlapping atoms. On the one hand, the first-principles calculations of these configurations usually suffer from difficulties to converge self-consistent field iterations. On the other hand, the energy of these configurations are usually prohibitively high, so they are statistically unfavorable, and hence are irrelevant to the accuracy of the final model. Therefore, the upper bound of trust level σ_{hi} is set to exclude the unphysical configurations from the labeling stage to prevent numerical difficulties in the first-principles calculations and to save the computational cost. A rule of thumb setting of σ_{hi} is 0.15 –0.30 eV/Å higher than σ_{lo}.
- $\sigma_{lo} \le \epsilon_t < \sigma_{hi}$. Based on the reasoning above, structures corresponding to this situation are selected as candidates for labeling.

3. Software

3.1. Overview

Implemented with Python, DP-GEN provides a user-friendly interface. The master process can be started via a single line of command:

```
dpgen run PARAM MACHINE
```

where the arguments PARAM and MACHINE are both the names of parameter files in the json format that specify the user's demands.

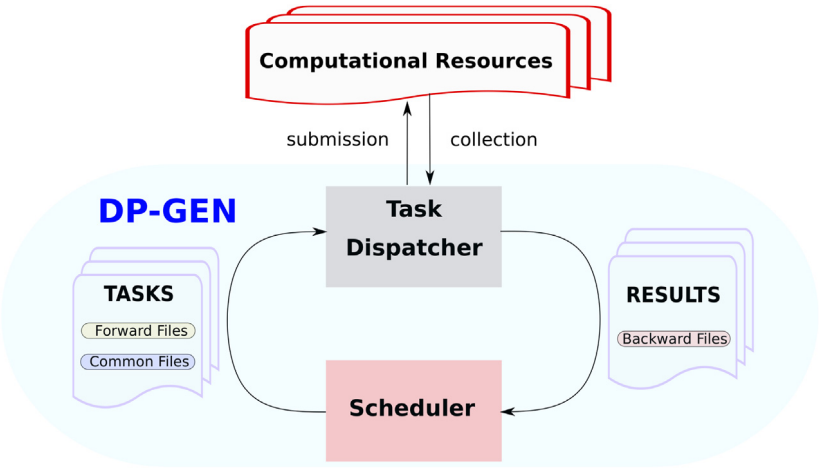

Fig. 2. (Color online) Overview of the structure of the DP-GEN package, which has two major modules, scheduler and task dispatcher. The scheduler prepares and sends the calculation tasks to the task dispatcher. By communicating with computational resources, the task dispatcher automatically handles job submission and the collection of the results. Results are then sent back to the scheduler, and the master process of DP-GEN enters the next step.

DP-GEN is composed of two major modules. First, DP-GEN serves as a scheduler, which follows the aforementioned concurrent learning scheme and generates computational tasks iteratively for the three steps: exploration, labeling and training. Second, DP-GEN serves as a task dispatcher, which receives tasks from the scheduler and automatically submits tasks to available computational resources, collects results when a task finishes, and sends the results back to the scheduler.

Fig. 2 shows the communication between the scheduler module and dispatcher module. The scheduler prepares and sends the calculation tasks to the task dispatcher, and in return, it receives results collected by the dispatcher. By communicating with the computational resources and using available job management tools, the task dispatcher automatically handles job submission and result collection. Typically, a task is composed of two kinds of files, forward files and common files. Forward files are specific for different tasks, while common files contain universal settings for all tasks. The results contained in backward files are then sent back from the dispatcher to the scheduler when the task is finished.

3.2. DP-GEN scheduler

The scheduler maintains the information flow between different steps of the concurrent learning procedure. It always works on the machine on which the master process of DP-GEN runs. To manipulate format transformations between data files generated by different software, the scheduler utilizes an auxiliary open-source Python package named `dpdata`, available at GitHub [19], developed by the authors. Details for the implementation of the three steps are as follows.

Exploration. At this moment, DP-GEN only supports the use of the LAMMPS package [14], which has to be interfaced with the DeePMD-kit package to perform DP-based MD (DPMD). The scheduler prepares exploration tasks, i.e., the DP models `graph.00x.pb`, input files `in.lmp`, and initial structures `conf.lmp` required by LAMMPS. In return, it collects the configurations sampled by LAMMPS and selects a subset, according to the criterion for the model deviation, for the labeling step.

The exploration process is based on a sampler, which produces abundant configurations and saves the trajectories in the `traj` folder with a predefined frequency. For each configuration, the model deviation is calculated, and the results are saved in `model_devi.out`.

The scheduler categorizes all the configurations sampled by the DP model into the three types according to the model deviations ($\sigma_{hi} \leq \epsilon_t$, $\epsilon_t < \sigma_{lo}$ and $\sigma_{lo} \leq \epsilon_t < \sigma_{hi}$). DP-GEN will then show users the distributions of configurations belonging to different types respectively in rest_failed.xxx.out, rest_accurate.xxx.out and candidate.xxx.out.

Labeling. In this step, DP-GEN currently supports VASP [20,21], Quantum-Espresso [22], Gaussian [23], and CP2K [24]. The scheduler prepares different input files according to the requirements of the different software. In return, it receives first-principles energies, forces, and/or virials, and adds them to the training data.

Selected configurations from the *exploration* step will be calculated using one of these software. DP-GEN allows and recommends users to take the whole control of their settings for *ab initio* calculations. The results can be analyzed by the scheduler and new labeled data will be added to the training dataset. It should be emphasized that the scheduler can handle the case where the expected convergence of the self-consistent-field calculation, adopted by many *ab initio* calculations, is not achieved. In this case, the results are considered unqualified and thus excluded from the training dataset.

Training. In this step, the DeePMD-kit package [10] is used for training. The scheduler prepares training tasks, i.e., the training data and several `input.json` files required by DeePMD-kit. In return, it generates several DP models for efficient exploration in the next step.

The key information provided by the training files `input.json` are the paths to the training data and the hyperparameters that define the network structure and the training scheme. Notice that the training files only differ by the random seeds that initialize the network parameters. In addition, when the training data are accumulated as DP-GEN proceeds, the scheduler adds the directories of the new data to `input.json`. After training, the resulting DP models, named as `graph.00x.pb` where `x` goes from 0 to $N_m - 1$, are collected by the scheduler.

To keep track of the DP-GEN process, the scheduler takes records of the different steps. Once the scheduler steps forward, it saves checkpoints in `record.dpgen`. Therefore, when a fatal failure occurs and user intervention is required, the scheduler can restart from the latest checkpoint automatically. The user can also retrieve to any previous step when he/she finds that the current results are problematic and wants to modify the parameters.

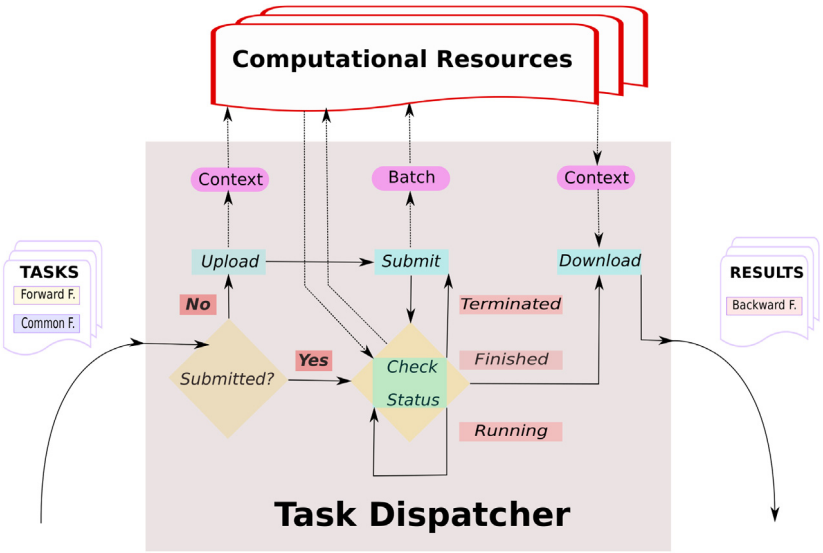

Fig. 3. (Color online) Schematic illustration of the task dispatcher of DP-GEN. The dispatcher receives the tasks to be calculated, and sends the results by communicating with the computational resources. When a task is sent into the dispatcher, the dispatcher first checks whether it has been submitted. If not, it will experience mainly four successive stages in the dispatcher. I. Upload. Through the class Context, the dispatcher sends all input files onto machines. II. Submit. The class Batch deals with providing adaptive scripts to designate proper computational resources for the task. Then, a job to calculate the task will be executed on the machine. III. Check_status. By querying the machines periodically, the dispatcher checks the status of all jobs, which can be running, terminated or finished. Running status leads to nothing but waiting for the next check. If a job is found terminated, it will be submitted again. Files in jobs with finished status will be downloaded. IV. Download. The class Context takes charge of transferring files containing valuable results back into the dispatcher. Till now, all the four stages have been fulfilled, and a calculation task has finished. Additionally, those tasks submitted before will enter the third stage check_status directly.

3.3. Task dispatcher

Since the number of tasks passed from the scheduler is enormous, it would be tedious and time-consuming to manually manage script submission and result collection. This motivates us to equip DP-GEN with an intelligent task dispatcher. In general, the dispatcher automatically uploads input files, submits scripts for the calculations, keeps maintaining the job queues, and collects results when a job finishes. The dispatcher shows its strength in handling not only various types of calculation tasks, but also accommodating different kinds of available computational resources.

A typical workflow of the dispatcher is presented in Fig. 3 and is composed of following functions:

Check submission. First of all, the dispatcher will check whether the tasks have been submitted or not. If not, the dispatcher will upload files and construct the submission script from scratch. This will form a queue composed of tasks to be executed. Otherwise, the dispatcher will recover the existing queue and collect results from those tasks that have finished, instead of submitting the scripts again. Users will benefit a lot from such a design, in the sense that repeated computations are avoided when restarting the DP-GEN master process.

File uploading and downloading. When the DP-GEN master process is running on the login node of an HPC, it shares the file system with computational nodes via a network file system. File uploading and downloading can be simply implemented by Python modules `os` and `shutil`. To enhance the I/O efficiency, instead of copying or removing files, the dispatcher operates on the symbolic links or directly moves files. When the DP-GEN master process is running on a machine other than the machines that perform the actual computations, files are transmitted via `ssh`. The DP-GEN provides a uniform interface for these two situations, so file transmission is adapted to the connection type and can be invoked easily. Moreover, new protocols for file transmission can be implemented easily.

Job submission. After uploading forward and common files onto the computational machines, DP-GEN generates scripts for executing the desired computational tasks. The job scripts are adaptive to different kinds of machines, including workstations, high performance clusters with a job scheduling system, and cloud machines:

- *Workstation.* This allows users to run DP-GEN on a single computational machine, such as a personal laptop. In this situation, DP-GEN prepares shell scripts which can be directly executed.
- *HPCs with a job scheduling system*, such as Slurm [25], PBS [26], and LSF [27]. The resources to execute a job require setting in the submission scripts, such as the number of CPU/GPUs and maximal execution wall-time, etc. DP-GEN provides an end-to-end interface for users, and transforms the settings in MACHINE file to the submission scripts, which can be accepted by the job scheduling system.
- *Cloud machines.* The script for cloud machines is similar to one for workstation. The only difference is that the DP-GEN dispatcher runs additional commands to launch or terminate machines before or after the job execution, respectively. These commands are compatible with the application programming interface (API) that the cloud machine service provides.

Job monitoring. Monitoring the status of each job submitted is of vital importance. The DP-GEN dispatcher is able to identify the job status and react correspondingly by communicating with all kinds of the machines introduced above. If a job is running, the dispatcher will do nothing. If a job is terminated, the dispatcher will try to resubmit the job up to three times. If the job still cannot be successfully executed, it is highly likely that the job settings are problematic, e.g., the parameters might be improper or the configurations might be unphysical. In this case, user intervention is required. When a task is finished, the dispatcher downloads the backward files and passes them to the scheduler. The task dispatcher accomplishes its mission after all tasks are finished, and then the scheduler will step forward.

At last, it should be emphasized that DP-GEN is fairly automatic, general, and robust. Once the input files are prepared and DP-GEN runs successfully, there is no need for extra labor.

4. Examples

In this section, we report the details of the process that we follow to generate a DP model for Cu with DP-GEN, and demonstrate its uniform accuracy when used to predict a broad range of properties.

4.1. Generation of the model

To perform DP-GEN, we used LAMMPS for exploration, VASP 5.4.4 for labeling, and DeePMD-kit for training. In total, 48 iterations were undertaken. Among a total number of 25 million configurations sampled in the exploration step, 7646 (0.03%) are selected for labeling.

Exploration To kick off the DP-GEN procedure, we start with relaxed fcc, hcp, and bcc structures. The exploration in the first iteration is essentially random. For each crystalline structure, a $2 \times 2 \times 2$ supercell is first relaxed and compressed uniformly with a scaling factor α ranging from 0.84 to 1.00. Then the atomic positions and cell vectors are randomly perturbed. The magnitude of perturbations is 0.01 Å for the atomic coordinates, and is 3% of the cell vector length for the simulation cell. For each crystal structure, 50 randomly perturbed structures, whose α are 1.00, and 10 randomly perturbed structures whose α range from 0.84 to 0.98, are then utilized to perform a 20-step canonical AIMD simulation at $T = 50$ K with VASP.

In later exploration steps, we also need to prepare initial structures for DPMD sampling. For bulk systems, we choose the perturbed structures whose α are 1.00 for the three crystal structures. For surface systems, we rigidly displace two crystalline halves along crystallographic directions (100), (110), (111) for fcc, and (100), (001), (110) for hcp structures. The magnitude of the vacuum slab thickness ranges from 0.5 Å to 9 Å. The strategy for the bulk system adopts the protocol introduced in Section 2. For the surface systems, there are four iterations in each temperature interval, and the canonical ensemble (NVT) is used to explore the configurations. More details of the exploration strategy can be found in Table A.1. The directories of the initial configurations and the parameters (e.g. ensemble, length of the trajectories, sampling frequency) of the MD simulations in each iteration are specified by `sys_configs` and `model_devi_jobs`, respectively. Another vital component for the selection of configurations is the trust levels. We set σ_{lo} and σ_{hi} to 0.05 eV/Å and 0.2 eV/Å, respectively:

```
"model_devi_f_trust_lo": 0.05,
"model_devi_f_trust_hi": 0.20,
```

Labeling The Perdew–Burke–Ernzerhof (PBE) generalized gradient approximation [28] is utilized. The kinetic energy cut-off for the plane wave [20,21] is set to 650 eV, and the K-points is set using the Monkhorst–Pack mesh [29] with the spacing $h_k = 0.1$ Å^{-1}. The self-consistent-field iteration will stop when the total energy and band structure energy differences between two consecutive steps are smaller than 10^{-6} eV. If the number of candidate configurations of a crystal structure is larger than 300 for a bulk system, they are randomly down-sampled to 300 before labeling in each iteration. For surface systems, the maximal number of candidates to be labeled is 100 for each structure. This can be controlled by the key `fp_task_max`.

Users can also conveniently designate the directories of general settings (`INCAR`) and pseudo-potentials (`POTCAR`) in `PARAM`.

Table 1
Summary and description of the tests used to validate the DP model for Cu.

Order	Test	Description
0	Equilibrium state	The atomization energy E_{am} and equilibrium volume per atom V_0 at 0 K
1	Equation of state	The energy along a given range of volume
2	Elasticity	The elastic constants compared with experimental, DFT and MEAM results
3	Vacancy	The formation energy of vacancy defect compared with experiment, DFT and MEAM results
4	Interstitial	The formation energy of self-interstitial point defect compared with DFT results
5	Surface energy	The (*hkl*) surface energy compared with DFT and MEAM results
6	Phonon bands	The phonon band structures compared with experimental and MEAM results
7	Stacking-fault energies	The (111) stacking-fault energy compared with DFT and MEAM results

```
"fp_pp_path":  "/path/to/POTCAR",
"fp_incar":    "/path/to/INCAR"
```

After the DP-GEN workflow finishes, a representative set of training data is generated. To properly obtain the energy of a single copper atom in the vacuum, we perform an additional DFT calculation for a single copper atom located in a box of 19 Å. The corresponding data is duplicated 200 times and added to the training dataset. Finally, a productive DP model is trained with 8,000,000 steps, with the learning rates exponentially decaying from 1.0×10^{-3} to 3.5×10^{-8}.

Training. In this work, the smooth version of the DP model is adopted [8]. The cut-off radius is set to 8 Å, and the inverse distance $1/r$ decays smoothly from 2 Å to 8 Å in order to remove the discontinuity introduced by the cut-off. The embedding network of size (25, 50, 100) follows a ResNet-like architecture [30]. The fitting network is composed of three layers, with each containing 240 nodes. The Adam stochastic gradient descent method [31] is utilized to train four models, with the only difference being their random seeds. Each model is trained with 400,000 gradient descent steps with an exponentially decaying learning rate from 1.0×10^{-3} to 3.5×10^{-8}. These settings for the training process are designated by the key `default_training_params` provided by the input file PARAM, which adopts the same interface with DeePMD-kit.

Machine settings. In MACHINE file, one can specify the settings according to his/her own environment. We provide an easy example for the training machine settings on a Slurm system:

```
"machine": {
  "batch":     "slurm",
  "hostname":   "localhost",
  "port":      22,
  "username":   "user",
  "work_path":   "/path/to/Cu/work"
},
"resources": {
  "numb_node":   1,
  "numb_gpu":    1,
  "task_per_node":4,
  "partition":   "GPU",
  "source_list":  ["/path/to/env"]
}
```

Settings in `machine` specify how to connect with the computational machine and how to transfer files. All tasks will be sent to and run in the directories work_path on the computational

Table 2
Summary of machine settings and average costs for each step in a single iteration.

Step	Machine type	Cost
Training	Single card of Tesla-P100 GPU	6 (cards * h)
Exploration	Single card of Tesla-P100 GPU	29 (cards * h)
Labeling	Intel(R) Xeon(R) Gold 5117 CPU @ 2.00 GHz with 28 cores[a]	4678 (cores * h)

[a]The cost per hour of one single Tesla-P100 card is nearly equal to that of 100 CPU cores which we utilize.

Table 3
Equilibrium properties of Cu: atomization energy E_{am}, equilibrium volume per atom V_0, vacancy formation energy E_{vf}, self-interstitial point formation energies E_{if} for octahedral interstitial (oh) and tetrahedral interstitial (th), independent elastic constants C_{11}, C_{12}, and C_{44}, bulk modulus B_V (Voigt), shear modulus G_V (Voigt), stacking fault energy γ_{sf}.

Cu	EXP	DFT[a]	DP[b]	MEAM
E_{am} (eV/atom)	−3.563[c]	−3.712	−3.7098(1)	−3.540
V_0 (Å^3/atom)[d]	11.65[e]	12.00	12.004(1)	11.65
E_{vf} (eV)	1.29[f]	1.020	0.99(1)	1.105
E_{if}(oh) (eV)		3.616	3.472(6)	3.136
E_{if}(th) (eV)		3.999	4.149(7)	4.604
C_{11} (GPa)	176.2[g]	171.74	173(2)	175.76
C_{12} (GPa)	124.9[g]	118.91	122(2)	124.09
C_{44} (GPa)	81.77[g]	81.59	75.7(3)	77.59
B_V (GPa)	142.0[g]	136.52	139(1)	141.31
G_V (GPa)	59.32[g]	58.32	55.8(5)	56.89
γ_{sf} (mJ/m^2)	41[h]	38.08	36(2)	72.7

[a]The DFT results are computed by the authors. While a smaller K-mesh spacing in DFT may lead to more converged results, we set the K-mesh spacing equal to 0.1 Å^{-1}, in order to be consistent with settings in the labeling step.
[b]The numbers in parentheses are standard deviations among 4 models in the last one digit.
[c]Ref. [32].
[d]Experiment value was extrapolated to absolute zero and corrected for zero-point vibrations; DFT, DP and MEAM results obtained at $T = 0$ K.
[e]Ref. [33].
[f]Ref. [34].
[g]Ref. [35].
[h]Ref. [36].

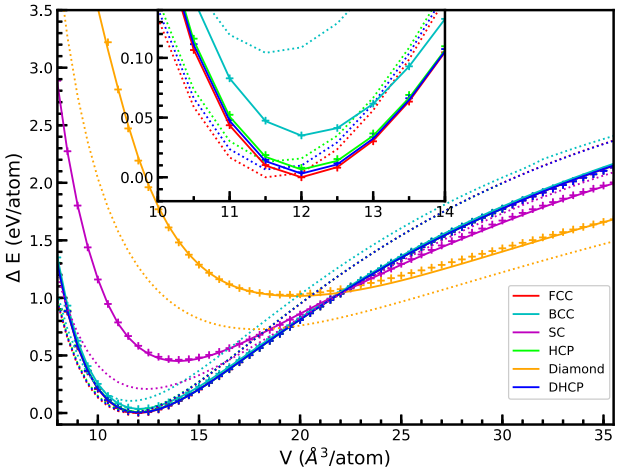

Fig. 4. EOS curves for Cu. Solid lines, dotted lines, and plus markers denote DP, MEAM, and DFT results, respectively. The energies of MEAM are shifted so that the MEAM energy of a stable fcc structure equals that given by DFT. DFT based relaxations fail for some hcp structures with volume per atom larger than 30 Å^3. Therefore, the corresponding DFT predictions are not shown. The diamond, sc, and dhcp structures are not explicitly included in the training data of the DP model.

machines. Reading the keyword `resources`, DP-GEN automatically requires for resources that users need among the machines,

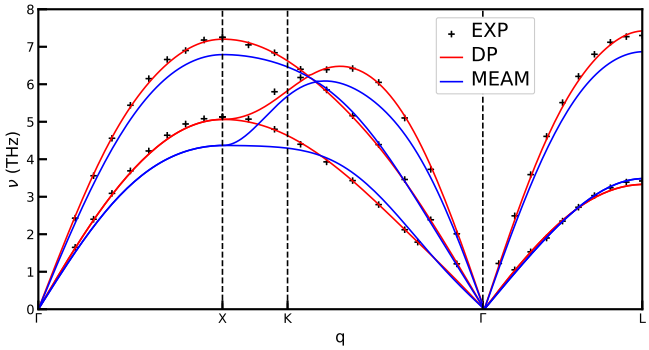

Fig. 5. The DP and MEAM results of phonon dispersion relations for Cu are calculated by phonopy [37] and its LAMMPS interface phonolammps [38] at $T = 49$ K with the supercell size being $10 \times 10 \times 10$. Here q denotes the wave number and ν the frequency.
Source: The experimental data are taken from Ref. [39].

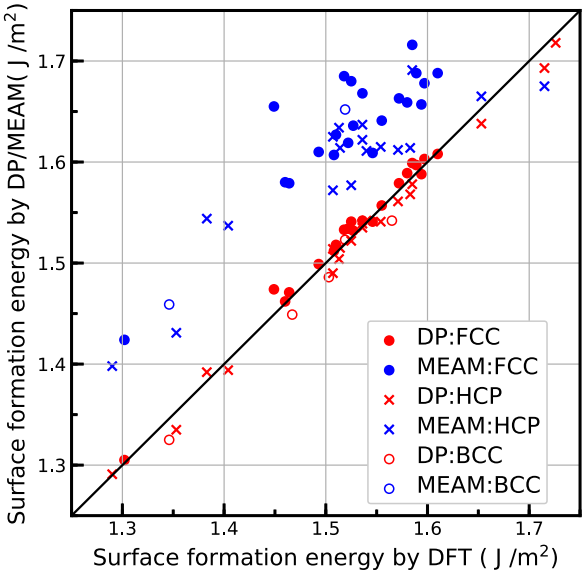

Fig. 6. Surface formation energies for fcc, hcp and bcc-lattices of Cu. All the nonequivalent surfaces with Miller index values smaller than 4 are taken into the consideration for the fcc lattice. For the hcp and bcc lattices, the results for the nonequivalent surfaces with Miller index values smaller than 3 are included.

such as the number of GPU cards and CPU cores, the prerequisite modules for the software that performs the calculation, etc.

Regarding the computational cost, we summarize the machine types and the average costs for each step in a single iteration in Table 2. It is observed that training requires relatively less computational resources than exploration and labeling. Labeling is still the most expensive part, and it could be relatively more expensive if we use a more complicated functional approximation or if we study a system with more valence electrons, since these will not affect the efficiency of the training and exploration steps.

4.2. Testing the model

To validate the performance of the DP model for pure copper, we test it on a wide range of properties, as summarized in Table 1. We compare the results of the DP model in predicting the important material properties with a state-of-the-art empirical FF model, obtained by the modified embedded atom method (MEAM) [40].

The equilibrium properties of Cu are presented in Table 3. These include the atomization energy and equilibrium volume

Table A.1
Exploration strategy for the copper system. For each iteration, we report the crystalline structure from which the initial structures derive, the number of DPMD simulations, the length of trajectories, the simulation temperatures, the statistical ensembles, and the percentages of candidates for labeling. For those simulations which adopt the NPT ensemble, 1, 10 100, 1000, 5000, 10,000, 20,000 and 50,000 Bar are set as the pressures.

Iter.	Crystal	#DPMD	Length (ps)	T (K)	Ensemble	Candidate Per (%)
0	FCC, HCP, BCC	600	2	50,135,271,407,543	NPT	8.29
1	FCC, HCP, BCC	600	2	50,135,271,407,543	NPT	0.00
2	FCC, HCP, BCC	1200	6	50,135,271,407,543	NPT	0.00
3	FCC, HCP, BCC	1200	6	50,135,271,407,543	NPT	0.00
4	FCC, HCP, BCC	1200	6	50,135,271,407,543	NPT	0.00
5	FCC, HCP, BCC	1200	6	50,135,271,407,543	NPT	0.00
6	FCC, HCP, BCC	2400	6	50,135,271,407,543	NPT	0.00
7	FCC, HCP, BCC	2400	6	50,135,271,407,543	NPT	0.00
8	FCC, HCP, BCC	600	2	678,814,950,1086,1221	NPT	5.45
9	FCC, HCP, BCC	600	2	678,814,950,1086,1221	NPT	0.00
10	FCC, HCP, BCC	1200	6	678,814,950,1086,1221	NPT	0.01
11	FCC, HCP, BCC	1200	6	678,814,950,1086,1221	NPT	0.01
12	FCC, HCP, BCC	1200	6	678,814,950,1086,1221	NPT	0.01
13	FCC, HCP, BCC	1200	6	678,814,950,1086,1221	NPT	0.00
14	FCC, HCP, BCC	2400	6	678,814,950,1086,1221	NPT	0.00
15	FCC, HCP, BCC	2400	6	678,814,950,1086,1221	NPT	0.00
16	FCC, HCP, BCC	600	2	1357,1493,1629,1765,1900	NPT	2.96
17	FCC, HCP, BCC	600	2	1357,1493,1629,1765,1900	NPT	0.00
18	FCC, HCP, BCC	1200	6	1357,1493,1629,1765,1900	NPT	0.01
19	FCC, HCP, BCC	1200	6	1357,1493,1629,1765,1900	NPT	0.01
20	FCC, HCP, BCC	1200	6	1357,1493,1629,1765,1900	NPT	0.01
21	FCC, HCP, BCC	1200	6	1357,1493,1629,1765,1900	NPT	0.01
22	FCC, HCP, BCC	2400	6	1357,1493,1629,1765,1900	NPT	0.00
23	FCC, HCP, BCC	2400	6	1357,1493,1629,1765,1900	NPT	0.00
24	FCC, HCP, BCC	600	2	2036,2172,2308,2443,2579	NPT	0.06
25	FCC, HCP, BCC	600	2	2036,2172,2308,2443,2579	NPT	0.04
26	FCC, HCP, BCC	1200	6	2036,2172,2308,2443,2579	NPT	0.02
27	FCC, HCP, BCC	1200	6	2036,2172,2308,2443,2579	NPT	0.08
28	FCC, HCP, BCC	1200	6	2036,2172,2308,2443,2579	NPT	0.00
29	FCC, HCP, BCC	1200	6	2036,2172,2308,2443,2579	NPT	0.00
30	FCC, HCP, BCC	2400	6	2036,2172,2308,2443,2579	NPT	0.01
31	FCC, HCP, BCC	2400	6	2036,2172,2308,2443,2579	NPT	0.00
32	FCC (Surf), HCP (Surf)	2400	2	50,135,271,407,543	NVT	53.97
33	FCC (Surf), HCP (Surf)	2400	2	50,135,271,407,543	NVT	0.16
34	FCC (Surf), HCP (Surf)	4800	6	50,135,271,407,543	NVT	0.04
35	FCC (Surf), HCP (Surf)	4800	6	50,135,271,407,543	NVT	0.00
36	FCC (Surf), HCP (Surf)	2400	2	678,814,950,1086,1221	NVT	0.01
37	FCC (Surf), HCP (Surf)	2400	2	678,814,950,1086,1221	NVT	0.16
38	FCC (Surf), HCP (Surf)	4800	6	678,814,950,1086,1221	NVT	0.02
39	FCC (Surf), HCP (Surf)	4800	6	678,814,950,1086,1221	NVT	0.01
40	FCC (Surf), HCP (Surf)	2400	2	1357,1493,1629,1765,1900	NVT	0.16
41	FCC (Surf), HCP (Surf)	2400	2	1357,1493,1629,1765,1900	NVT	0.05
42	FCC (Surf), HCP (Surf)	4800	6	1357,1493,1629,1765,1900	NVT	0.22
43	FCC (Surf), HCP (Surf)	4800	6	1357,1493,1629,1765,1900	NVT	0.04
44	FCC (Surf), HCP (Surf)	2400	2	2036,2172,2308,2443,2579	NVT	0.05
45	FCC (Surf), HCP (Surf)	2400	2	2036,2172,2308,2443,2579	NVT	0.08
46	FCC (Surf), HCP (Surf)	4800	6	2036,2172,2308,2443,2579	NVT	0.03
47	FCC (Surf), HCP (Surf)	4800	6	2036,2172,2308,2443,2579	NVT	0.24

per atom, defect formation energies, elastic constants and moduli, and stacking-fault energies. The defect formation energy is defined as $E_{\mathrm{df}} = E_{\mathrm{d}}(N_{\mathrm{d}}) - N_{\mathrm{d}}E_0$. Here, $d = v(i)$ indicates vacancy (self-interstitial) defects, $E_{\mathrm{d}}(N_{\mathrm{d}})$ denotes the relaxed energy of a defective structure with N_{d} atoms and E_0 denotes the energy per atom of the corresponding ideal crystal at $T = 0$ K. Besides, we replicate the primitive fcc cell $3 \times 3 \times 3$ times to generate a supercell, and use it to compute the defect formation energies. For all the properties listed in Table 3, the DP predictions agree well with DFT and/or experiments.

The predictions via DFT, DP, and MEAM for the equation of state (EOS) are presented in Fig. 4. DP reproduces well the DFT results for all the standard crystalline structures considered here, i.e., fcc, hcp, double hexagonal close-packed (dhcp), bcc, simple cubic (sc) and diamond. It is worth noting that the diamond, sc and dhcp structures are not explicitly explored by the DP-GEN scheme, i.e., the initial training data and the initial structures for exploration do not contain these crystal structures. Nevertheless, DP still achieves a satisfactory accuracy in the EOS test on these three structures. In comparison, although MEAM performs well for fcc, hcp and dhcp structures near the energy minimum, it shows large deviations when predicting the EOS for sc, diamond and bcc structures. We also report the DP and MEAM predictions for the phonon dispersion relations as well as experimental results. As shown in Fig. 5, DP results agree very well with the experiment and are significantly better than MEAM predictions.

Finally, we consider the surface formation energy $E_{\mathrm{sf}}((hkl))$, which describes the energy needed to create a surface with Miller indices (hkl) for a given crystal, and is defined by $E_{\mathrm{sf}}((hkl)) = \frac{1}{2A}[E_{\mathrm{s}}((hkl)) - N_{\mathrm{s}}E_0]$. Here $E_{\mathrm{s}}((hkl))$ and N_{s} denote the energy and number of atoms of the relaxed surface structure with Miller indices (hkl). A denotes the surface area. We enumerate all the nonequivalent surfaces with the Miller indices smaller than 4 for the fcc lattice and smaller than 3 for the hcp and bcc lattices. As shown in Fig. 6, despite the lack of any explicitly labeled data for bcc surfaces in the training dataset, the surface formation energies predicted by DP are close to those by DFT, and are significantly better than those predicted by MEAM.

Table A.2
The predictions of surface formation energies E_{sf} (in J/m^2)for fcc, hcp and bcc-lattices and their standard deviations $\sigma(E_{sf})$ among 4 models.

fcc	Miller indices ($h.k.l$)	E^1_{sf}	E^2_{sf}	E^3_{sf}	E^4_{sf}	$\sigma(E_{sf})$
000	1.1.1	1.302	1.306	1.298	1.301	0.003
001	3.3.2	1.488	1.492	1.496	1.498	0.004
002	3.3.1	1.532	1.537	1.545	1.544	0.005
003	1.1.0	1.476	1.479	1.483	1.482	0.003
004	3.3.−1	1.526	1.530	1.536	1.536	0.004
005	3.3.−2	1.536	1.540	1.545	1.544	0.004
006	1.1.−1	1.531	1.533	1.535	1.534	0.001
007	3.2.2	1.505	1.511	1.512	1.513	0.003
008	3.2.1	1.611	1.612	1.610	1.613	0.001
009	3.2.−1	1.597	1.600	1.603	1.605	0.003
010	3.2.−2	1.598	1.600	1.600	1.602	0.001
011	3.2.−3	1.581	1.581	1.580	1.584	0.002
012	3.1.−1	1.591	1.592	1.591	1.594	0.001
013	3.1.−2	1.607	1.608	1.605	1.610	0.002
014	3.1.−3	1.591	1.592	1.590	1.594	0.001
015	3.0.−1	1.459	1.466	1.463	1.470	0.004
016	3.0.−2	1.531	1.535	1.534	1.536	0.002
017	1.0.−1	1.545	1.545	1.544	1.544	0.000
018	3.−1.−1	1.450	1.455	1.454	1.459	0.003
019	3.−1.−2	1.553	1.555	1.556	1.559	0.002
020	3.−2.−2	1.511	1.514	1.517	1.518	0.003
hcp	Miller indices ($h.k.l$)	E^1_{sf}	E^2_{sf}	E^3_{sf}	E^4_{sf}	$\sigma(E_{sf})$
000	1.1.1	1.688	1.694	1.700	1.698	0.005
001	1.1.1	1.387	1.388	1.388	1.386	0.001
002	2.2.1	1.389	1.392	1.395	1.397	0.003
003	2.2.1	1.573	1.576	1.582	1.582	0.004
004	1.1.0	1.713	1.714	1.725	1.721	0.005
005	1.1.0	1.329	1.327	1.328	1.327	0.001
006	2.1.2	1.510	1.509	1.513	1.510	0.001
007	2.1.1	1.562	1.560	1.563	1.561	0.001
008	2.1.0	1.517	1.515	1.518	1.517	0.001
009	2.−1.2	1.534	1.532	1.538	1.537	0.002
010	2.−1.2	1.530	1.531	1.539	1.538	0.004
011	2.−1.1	1.535	1.536	1.541	1.542	0.003
012	2.−1.1	1.536	1.535	1.540	1.539	0.002
013	2.−1.0	1.485	1.488	1.489	1.492	0.002
014	2.−1.0	1.633	1.630	1.631	1.633	0.001
015	2.−2.1	1.556	1.555	1.556	1.556	0.000
016	1.1.2	1.508	1.508	1.517	1.514	0.004
017	1.1.2	1.499	1.504	1.509	1.511	0.005
018	0.0.1	1.285	1.285	1.278	1.281	0.003
bcc	Miller indices ($h.k.l$)	E^1_{sf}	E^2_{sf}	E^3_{sf}	E^4_{sf}	$\sigma(E_{sf})$
000	1.1.1	1.541	1.537	1.544	1.542	0.003
001	2.2.1	1.484	1.486	1.494	1.493	0.004
002	1.1.0	1.323	1.333	1.332	1.327	0.004
003	2.1.1	1.214	1.208	1.214	1.212	0.002
004	2.1.0	1.447	1.45	1.459	1.457	0.005
005	1.0.0	1.521	1.525	1.535	1.53	0.005

5. Conclusion

In this paper, we introduced the software platform DP-GEN. We described its implementation and reported the details when used to generate a general purpose PES model for Cu. We expect DP-GEN to be a scalable and flexible platform. The three steps, exploration, training, and labeling, which are controlled by the scheduler, are separate and highly modularized. Therefore, developers will spend a minimal amount of effort to incorporate novel functionalities. For example, DP-GEN can easily be extended to include a different first-principles code, for which typically only file conversion and job submission scripts are needed. Furthermore, a list of sampling techniques may be added to the exploration step.

Provided the ability of the DP model to describe various systems, such as organic molecules, metals, semiconductors, and insulators, we expect DP-GEN to be widely used in different molecular and materials science applications. Such applications are not limited to the generation of general purpose PES models, but also include the investigation of specific problems.

The DP-GEN workflow can be made applicable to different software and operating systems. In particular, it can be easily implemented on popular cloud machines such as the Amazon Web Services (AWS) cloud platform [41]. Moreover, there have been significant efforts to build automatic interactive platforms for computational science. Among these efforts, the AiiDA package [42] has become very promising for creating social ecosystems to disseminate codes, data, and scientific workflows. To connect DP-GEN with popular general-purpose open-source platforms is on our to-do list. Above all, we expect that users of DP-GEN will be embraced with an optimal solution in terms of both efficiency and cost when performing atomic and molecular simulations.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors thank Marcos F. Calegari Andrade, Hsin-Yu Ko, Jianxing Huang, Yunpei Liu, Mengchao Shi, Fengbo Yuan, and Yongbin Zhuang for helps and discussions. We are grateful for computing time provided by the TIGRESS High Performance Computer Center at Princeton University, the High-performance Computing Platform of Peking University, and the Beijing Institute of Big Data Research. The work of L. Z. and W. E was supported in part by a gift from iFlytek to Princeton University, USA, the ONR, USA grant N00014-13-1-0338, and the Center Chemistry in Solution and at Interfaces (CSI) funded by the DOE, USA Award DE-SC001934. The work of Han Wang is supported by the National Science Foundation of China under Grant No. 11871110, the National Key Research and Development Program of China under Grant Nos. 2016YFB0201200 and 2016YFB0201203, and Beijing Academy of Artificial Intelligence (BAAI), China. The work of J. Z. is partially supported by National Innovation and Entrepreneurship Training Program for Undergraduate, China (201910269080).

Appendix A. Details of the exploration strategy and more numerical results

See Tables A.1 and A.2.

Appendix B. Details of the error indicator

The error indicator ϵ_t recalled as

$$\epsilon_t = \max_i \sqrt{\langle \|F_{w,i}(\mathcal{R}_t) - \langle F_{w,i}(\mathcal{R}_t)\rangle\|^2\rangle} \tag{B.1}$$

estimates the upper bound of the atomic force error of a structure. We notice that the first-principles calculation is for the whole periodic structure, and it is non-trivial to cut-off the atoms with small force error and to label only the atoms with large force error. Therefore, if at least one of the atoms in the structure is of low accuracy, the whole structure is selected as candidate for the labeling stage.

We also notice that, the error indicator always underestimates the real force error. Let $\tilde{F}_i(\mathcal{R}_t)$ denote the first-principles force on the atom with index i. For the configuration $\mathcal{R}_t$, the real error

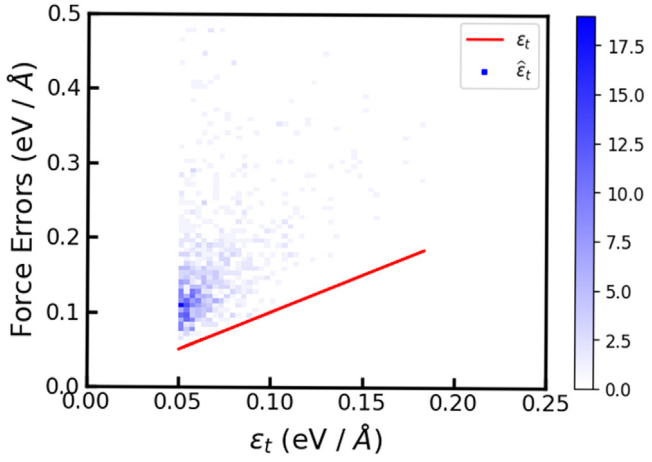

Fig. B.1. Comparisons between the error indicator ϵ_t (red solid lines) and force prediction errors $\widehat{\epsilon}_t$ (grids) at Iteration 8. There are 4.3% points whose force prediction errors are above 0.5 eV/Å. The color bar scale is tuned for a proper density comparison.

defined by the maximal root-mean-squared real atomic force error is given by

$$\widehat{\epsilon}_t = \max_i \sqrt{\langle \|F_{w,i}(\mathcal{R}_t) - \tilde{F}_i(\mathcal{R}_t)\|^2 \rangle} \tag{B.2}$$

Notice that

$$\begin{aligned} &\langle \|F_{w,i}(\mathcal{R}_t) - \tilde{F}_i(\mathcal{R}_t)\|^2 \rangle \\ &= \langle \|F_{w,i}(\mathcal{R}_t) - \langle F_{w,i}(\mathcal{R}_t)\rangle + \langle F_{w,i}(\mathcal{R}_t)\rangle - \tilde{F}_i(\mathcal{R}_t)\|^2 \rangle \\ &= \langle \|F_{w,i}(\mathcal{R}_t) - \langle F_{w,i}(\mathcal{R}_t)\rangle\|^2 \rangle + \|\langle F_{w,i}(\mathcal{R}_t)\rangle - \tilde{F}_i(\mathcal{R}_t)\|^2, \end{aligned} \tag{B.3}$$

and the second item $\|\langle F_{w,i}(\mathcal{R}_t)\rangle - \tilde{F}_i(\mathcal{R}_t)\|^2$ is non-negative, then the real error $\widehat{\epsilon}_t$ is always larger than the error indicator ϵ_t, unless the ensemble averaged force $\langle F_{w,i}(\mathcal{R}_t)\rangle$ is an unbiased estimate of the first-principles force $\tilde{F}_i(\mathcal{R}_t)$.

As an example, we compare in Fig. B.1 the error indicator and the force prediction error at iteration 8 of the DP-GEN process of Copper. It is shown that (1) the error indicator (plotted by the red line) is always smaller than the force prediction error; and (2) a larger error indicator implies a larger real error; and (3) in a few extreme cases, the real error can be 10 times larger than the estimated error, so the error indicator is not sharp. Therefore, at the end of the exploration stage of each iteration, the error indicator may miss some structures of large real error, but all structures selected by the error indicator has a error that is truly above the trust level σ_{lo}, and should be proposed for first-principles calculation.

References

[1] W. Kohn, L.J. Sham, Phys. Rev. 140 (4A) (1965) A1133.
[2] R. Car, M. Parrinello, Phys. Rev. Lett. 55 (22) (1985) 2471.
[3] D. Marx, J. Hutter, Ab Initio Molecular Dynamics: Basic Theory and Advanced Methods, Cambridge University Press, 2009.
[4] A.P. Bartók, M.C. Payne, R. Kondor, G. Csányi, Phys. Rev. Lett. 104 (13) (2010) 136403.
[5] J. Behler, M. Parrinello, Phys. Rev. Lett. 98 (14) (2007) 146401.
[6] J. Han, L. Zhang, R. Car, W. E, Commun. Comput. Phys. 23 (3) (2018) 629–639.
[7] L. Zhang, J. Han, H. Wang, R. Car, W. E, Phys. Rev. Lett. 120 (2018) 143001.
[8] L. Zhang, J. Han, H. Wang, W. Saidi, R. Car, W. E, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 31, Curran Associates, Inc., 2018, pp. 4441–4451.
[9] N. Artrith, A. Urban, Comput. Mater. Sci. 114 (2016) 135–150.
[10] H. Wang, L. Zhang, J. Han, W. E., Comput. Phys. Comm. 228 (2018) 178–184.
[11] K. Schutt, P. Kessel, M. Gastegger, K. Nicoli, A. Tkatchenko, K.-R. Müller, J. Chem. Theory Comput. 15 (1) (2018) 448–455.
[12] K. Yao, J.E. Herr, D.W. Toth, R. Mckintyre, J. Parkhill, Chem. Sci. 9 (8) (2018) 2261–2269.
[13] 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. Mane, 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. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016, arXiv:1603.04467 [cs.DC].
[14] S. Plimpton, J. Comput. Phys. 117 (1) (1995) 1–19.
[15] M. Ceriotti, J. More, D.E. Manolopoulos, Comput. Phys. Comm. 185 (2014) 1019–1026.
[16] E.V. Podryabinkin, A.V. Shapeev, Comput. Mater. Sci. 140 (2017) 171–180.
[17] J.S. Smith, B. Nebgen, N. Lubbers, O. Isayev, A.E. Roitberg, J. Chem. Phys. 148 (24) (2018) 241733.
[18] L. Zhang, D.-Y. Lin, H. Wang, R. Car, W. E, Phys. Rev. Mater. 3 (2) (2019) 023804.
[19] See https://github.com/deepmodeling/dpdata for code implementation.
[20] G. Kresse, J. Furthmüller, Comput. Mater. Sci. 6 (1) (1996) 15–50.
[21] G. Kresse, J. Furthmüller, Phys. Rev. B 54 (16) (1996) 11169.
[22] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M.B. Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, et al., J. Phys.: Condens. Matter 29 (46) (2017) 465901.
[23] M.J. Frisch, G.W. Trucks, H.B. Schlegel, G.E. Scuseria, M.A. Robb, J.R. Cheeseman, G. Scalmani, V. Barone, G.A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A.V. Marenich, J. Bloino, B.G. Janesko, R. Gomperts, B. Mennucci, H.P. Hratchian, J.V. Ortiz, A.F. Izmaylov, J.L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V.G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J.A. Montgomery, J.E. Peralta, F. Ogliaro, M.J. Bearpark, J.J. Heyd, E.N. Brothers, K.N. Kudin, V.N. Staroverov, T.A. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A.P. Rendell, J.C. Burant, S.S. Iyengar, J. Tomasi, M. Cossi, J.M. Millam, M. Klene, C. Adamo, R. Cammi, J.W. Ochterski, R.L. Martin, K. Morokuma, O. Farkas, J.B. Foresman, D.J. Fox, Gaussian16 Revision C.01, Gaussian Inc., Wallingford CT, 2016.
[24] J. Hutter, M. Iannuzzi, F. Schiffmann, J. Vandevondele, Wiley Interdiscip. Rev. Comput. Mol. Sci. 4 (1) (2014) 15–25.
[25] https://slurm.schedmd.com/.
[26] https://www.pbsworks.com.
[27] https://www.ibm.com/support/knowledgecenter/en/SSETD4/product_welcome_platform_lsf.html.
[28] J.P. Perdew, K. Burke, M. Ernzerhof, Phys. Rev. Lett. 77 (1996) 3865–3868.
[29] H.J. Monkhorst, J.D. Pack, Phys. Rev. B 13 (12) (1976) 5188.
[30] K. He, X. Zhang, S. Ren, J. Sun, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
[31] D. Kingma, J. Ba, Adam: a method for stochastic optimization, in: Proceedings of the International Conference on Learning Representations (ICLR), 2015.
[32] V.A. Medvedev, J. Cox, D.D. Wagman, CODATA Key Values for Thermodynamics, Hemisphere Publishing Corporation, New York, 1989.
[33] K. Lejaeghere, V. Van Speybroeck, G. Van Oost, S. Cottenier, Crit. Rev. Solid State Mater. Sci. 39 (1) (2014) 1–24.
[34] W. Triftshäuser, Phys. Rev. B 12 (11) (1975) 4634.
[35] W. Overton Jr, J. Gaffney, Phys. Rev. 98 (4) (1955) 969.
[36] W. Stobbs, C. Sworn, Phil. Mag. 24 (192) (1971) 1365–1381.
[37] A. Togo, I. Tanaka, Scr. Mater. 108 (2015) 1–5.
[38] See LAMMPS interface in https://github.com/abelcarreras/phonolammps for phonon calculations using phonopy.
[39] R. Nicklow, G. Gilat, H. Smith, L. Raubenheimer, M. Wilkinson, Phys. Rev. 164 (3) (1967) 922.
[40] M.I. Baskes, Phys. Rev. B 46 (5) (1992) 2727.
[41] https://aws.amazon.com.
[42] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, B. Kozinsky, Comput. Mater. Sci. 111 (2016) 218–230.